


JAVA Programming
MCA 109

UNIT 4


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.1



Learning Objectives

- **JDBC:** Introduction to DBMS & RDBMS, DBC API, JDBC Application Architecture, Obtaining a Connection, JDBC Models: Two Tier and Three Tier Model, Result Set, Prepared Statement, Callable Statement.
- **Java 8 Concepts:** Default and Functional Interfaces, Lambda Expression, Java stream API and Pipelines, Try with Resources, Java 8 Memory optimization.
- **RMI (Remote Method Invocation):** Introduction, Steps in creating a Remote Object, Generating Stub & Skeleton, RMI Architecture, RMI packages.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.2



JDBC

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.3

JDBC

- Java JDBC is a **java API** to **connect** and **execute query** with the **database**.
- JDBC API uses **jdbc drivers** to connect with the database.

```
graph LR; API((JDBC API)) --> App[Java Application]; App --> Driver((JDBC Driver)); Driver --> DB[(Database)];
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.4

API


- API (**Application programming interface**) is a document that contains **description** of all the features of a product or software.
- It represents **classes** and **interfaces** that software programs can follow to communicate with each other.
- An API can be created for applications, libraries, operating systems, etc.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.5


Need for JDBC

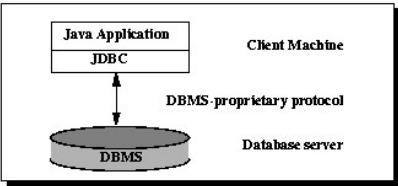
- Before JDBC, **ODBC API** was the database API to connect and execute query with the database.
- But, ODBC API uses ODBC driver which is written in **C language** (i.e. **platform dependent and insecure**).
- That is why Java has defined its own API (JDBC API) that uses **JDBC drivers (written in Java language)**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.6


 **JDBC Architecture**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.7

 **JDBC Two Tier Model**

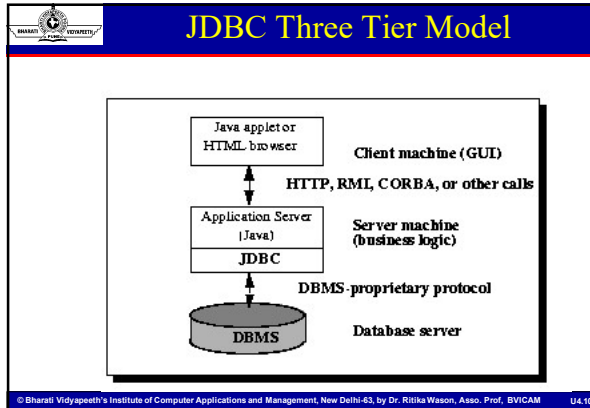


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.8

 **JDBC Two Tier Model**

- In the two-tier model, a **Java application** talks **directly** to the **data source**.
- This requires a **JDBC driver** that can communicate with the particular data source being accessed.
- A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user.
- The network can be an intranet or it can be the Internet.
- The data source may be located on **another machine** to which the user is connected via a **network**. This is referred to as a **client/server configuration**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.9



JDBC Three Tier Model

- In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source.
- The data source processes the commands and sends the results back to the middle tier, which then sends them to the user.
- Middle tier makes it possible to **maintain control** over access and the kinds of updates that can be made to corporate data.
- Another advantage is that it **simplifies the deployment of applications**. Finally, in many cases, the three-tier architecture can provide **performance advantages**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.11

JDBC Drivers

- JDBC Driver is a **software component** that enables java application to **interact** with the database.
- There are 4 types of JDBC drivers:
 - ✓ JDBC-ODBC bridge driver
 - ✓ Native-API driver (partially java driver)
 - ✓ Network Protocol driver (fully java driver)
 - ✓ Thin driver (fully java driver)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.12

JDBC Drivers

1. JDBC-ODBC Bridge

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of **thin driver**.

Figure- JDBC-ODBC Bridge Driver

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.13

JDBC Drivers

2. Native API Driver

The Native API driver uses the **client-side libraries** of the database. The driver converts JDBC method calls into **native calls** of the database API. It is not written entirely in java.

Figure- Native API Driver

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.14

JDBC Drivers

3. Network Protocol Driver

The Network Protocol driver uses **middleware** (application server) that converts JDBC calls directly or indirectly into the **vendor-specific database protocol**. It is fully written in java.

Figure- Network Protocol Driver

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.15

JDBC Drivers

4. Thin Driver

The thin driver converts JDBC calls **directly** into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

The diagram illustrates the Thin Driver architecture. On the left, a dashed box labeled 'Client Machine' contains three components: 'Jdbc API' (represented by a cylinder), 'Java Application' (represented by a rectangle), and 'Thin driver' (represented by a cylinder). Arrows indicate the flow of data: from 'Java Application' to 'Jdbc API', from 'Java Application' to 'Thin driver', and from 'Thin driver' to 'Database' (represented by a cylinder) located outside the client machine. Below the diagram is the caption 'Figure- Thin Driver'.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.16

Connection to Database


- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.17

DriverManager Class


- The DriverManager class acts as an **interface** between user and drivers.
- It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver()**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.18

 **Connection Interface**


- A Connection is the **session** between **java application** and **database**.
- The Connection interface is a factory of Statement, **PreparedStatement**, and **DatabaseMetaData** i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.
- The Connection interface provide many methods for transaction management like **commit()**, **rollback()** etc.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.19

 **Statement Interface**

- The **Statement interface** provides **methods** to execute **queries** with the database.
- The statement interface is a **factory** of **ResultSet** i.e. it provides factory method to get the object of ResultSet.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.20

 **ResultSet Interface**

- The object of ResultSet maintains a **cursor** pointing to a row of a table.
- Initially, cursor points to before the first row.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.21

PreparedStatement Interface

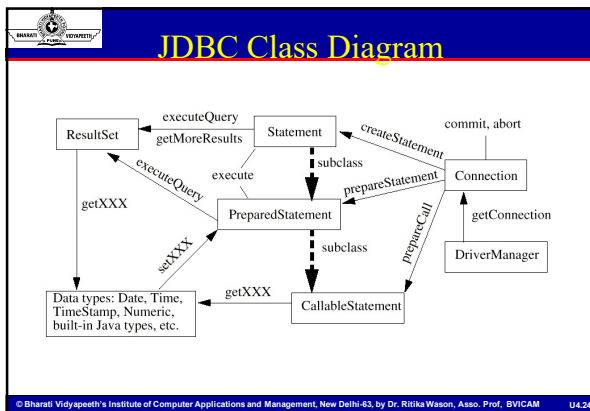
- The PreparedStatement interface is a **sub-interface** of **Statement**.
- It is used to **execute parameterized query**.
- Let's see the example of parameterized query:
 - ✓ `String sql="insert into emp values(?,?,?)";`


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.22

CallableStatement Interface


- CallableStatement interface is used to call the **stored procedures and functions**.
- We can have **business logic** on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.23




 **RMI**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.25

 **Remote Method Invocation**

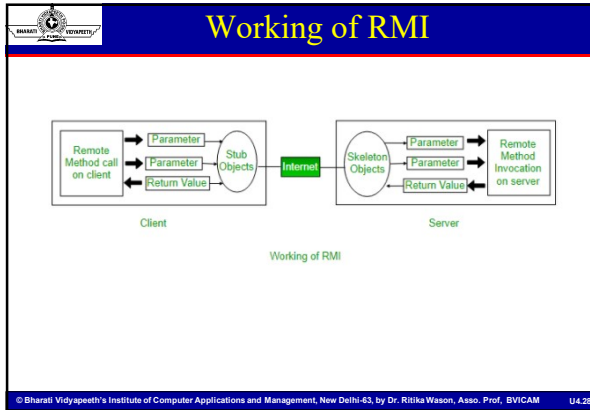
- Remote Method Invocation (RMI) is an API which allows an **object** to **invoke a method** on an **object** that exists in **another address space**, which could be on the **same machine** or on a **remote machine**.
- Through RMI, object running in a JVM present on a computer (**Client side**) can invoke methods on an object present in another JVM (**Server side**).
- RMI creates a **public remote server object** that enables client and server side communications through simple **method calls** on the server object.

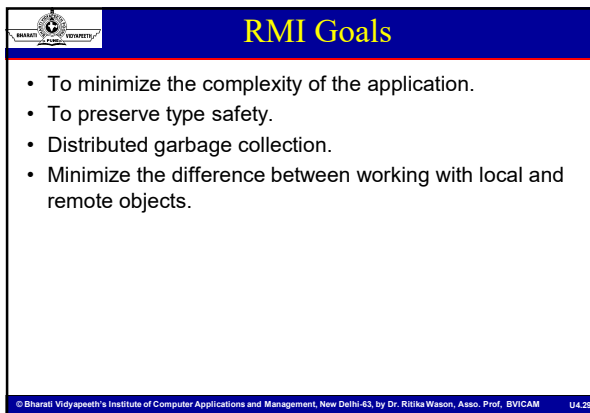
© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.26

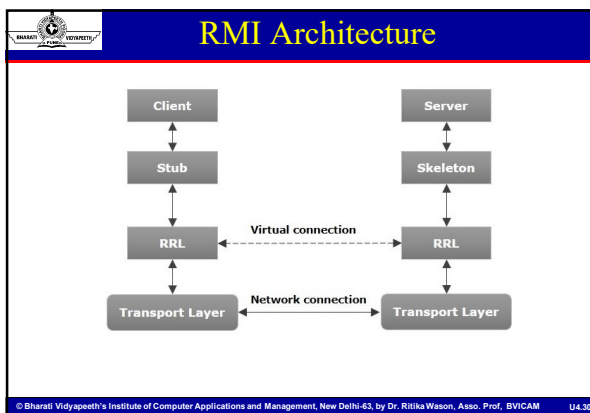
 **Working of RMI**


- The communication between client and server is handled by using **two intermediate objects**:
 - ✓ **Stub object** (on client side)
 - ✓ **Skeleton object** (on server side).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.27








 **RMI Architecture**


- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** – This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote objec

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.31

 **Stub Object**


- The stub object on the client machine builds an **information block** and sends this information to the server.
- The block consists of
 - ✓ An **identifier** of the remote object to be used
 - ✓ **Method name** which is to be invoked
 - ✓ **Parameters** to the remote JVM

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.32

 **Skeleton Object**


- The skeleton object passes the **request** from the stub object to the remote object.
- It performs following tasks
 - ✓ It calls the **desired method** on the **real object** present on the server.
 - ✓ It **forwards the parameters** received from the stub object to the method.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.33

 **Marshalling**


- Whenever a client invokes a method that **accepts parameters** on a remote object, the parameters are **bundled** into a **message** before being sent over the network.
- These parameters may be of primitive type or objects.
- In case of **primitive type**, the parameters are put together and a header is attached to it.
- In case the parameters are **objects**, then they are **serialized**. This process is known as **marshalling**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.34

 **Un-marshalling**

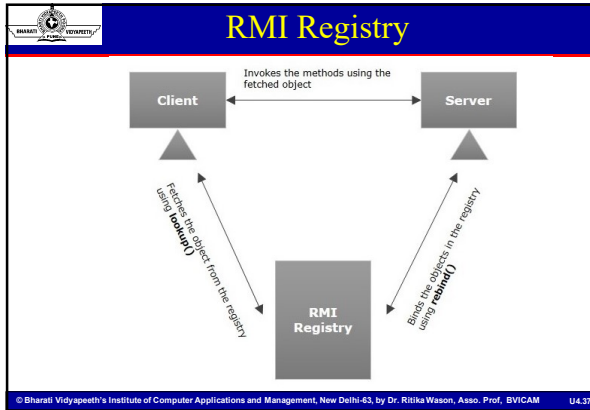
- At the server side, the packed parameters are **unbundled** and then the **required method** is invoked. This process is known as **unmarshalling**.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.35

 **RMI Registry**

- RMI registry is a **namespace** on which all **server objects are placed**.
- Each time the server creates an object, it registers this object with the RMIregistry using **bind()** or **reBind()** methods).
- These are registered using a unique name known as **bind name**.
- To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.36




Implementing RMI

- Defining a **remote interface**
- **Implementing** the remote interface
- Creating **Stub** and **Skeleton objects** from the implementation class using rmic (**rmi compiler**)
- Start the **rmi registry**
- Create and execute the **server application** program
- Create and execute the **client application** program.

Implementing RMI


Step 1: Defining the remote interface

The first thing to do is to create an interface which will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

 **Implementing RMI**

Step 1: Defining the remote interface
The first thing to do is to create an interface which will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.40

 **Implementing RMI**

Step 2: Implementing the remote interface

- The next step is to implement the remote interface. To implement the remote interface, the class should extend to UnicastRemoteObject class of java.rmi package. Also, a default constructor needs to be created to throw the java.rmi.RemoteException from its parent constructor in class.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.41

 **Implementing RMI**

Step 3: Creating Stub and Skeleton objects from the implementation class using rmic
The **rmic** tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects.
Its prototype is

rmic classname.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.42

 **Implementing RMI**

STEP 4: Start the `rmiregistry`
Start the registry service by issuing the following command at the command prompt

```
start rmiregistry
```


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.43

 **Implementing RMI**

STEP 5: Create and execute the `server application program`
The next step is to create the server application program and execute it on a separate command prompt.


- The server program uses `createRegistry` method of `LocateRegistry` class to create `rmiregistry` within the server JVM with the port number passed as argument.
- The `rebind` method of `Naming` class is used to bind the remote object to the new name.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.44


 **Implementing RMI**

Step 6: Create and execute the `client application program`
The last step is to create the client application program and execute it on a separate command prompt .The `lookup` method of `Naming` class is used to get the reference of the `Stub` object.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.45


 **Java 8 Concepts**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.46

 **Default and Functional Interfaces**


- Functional interfaces were introduced as part of Java 8. Implemented using the annotation called `@FunctionalInterface`. It ensures that the interface should have only one abstract method.
- The usage of the `abstract` keyword is optional as the method defined inside the interface is by default `abstract`. It is important to note that a functional interface can have multiple default methods (it can be said concrete methods which are default), but only one abstract method.
- The default method has been introduced in interface so that a new method can be appended in the class without affecting the implementing class of the existing interfaces. Prior to Java 8, the implementing class of an interface had to implement all the abstract methods defined in the interface.
- The functional interface has been introduced in Java 8 to support the lambda expression. On the other hand, it can be said lambda expression is the instance of a functional interface.
- Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.47

 **Example**

```
@FunctionalInterface
interface sayable{
    void say(String msg);
}
public class FunctionalInterfaceExample implements sayable{
    public void say(String msg){
        System.out.println(msg);
    }
    public static void main(String[] args) {
        FunctionalInterfaceExample fie = new FunctionalInterfaceExample();
        fie.say("Hello");
    }
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.48




Lambda Expression

- A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

The simplest lambda expression contains a single parameter:
parameter -> expression
 To use more than one parameter, wrap them in parentheses.
(parameter,parameter) -> expression

Expressions are limited. They have to immediately return a value, and they cannot contain variables, assignments or statements such as **if** or **for**. In order to do more complex operations, a code block can be used with curly braces. If the lambda expression needs to return a value, then the code block should have a **return** statement.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.49



Example

- `import java.util.ArrayList;`
- `public class Main {`
- `public static void main(String[] args) {`
- `ArrayList<Integer> numbers = new ArrayList<Integer>();`
- `numbers.add(5);`
- `numbers.add(9);`
- `numbers.add(8);`
- `numbers.add(1);`
- `numbers.forEach((n) -> { System.out.println(n); });`


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.50



java.stream API & Pipelines

- First of all, Java 8 Streams should not be confused with Java I/O streams (ex: FileInputStream etc); these have very little to do with each other.
- Simply put, streams are wrappers around a data source, allowing us to operate with that data source and making bulk processing convenient and fast.
- A stream does not store data and, in that sense, is not a data structure. It also never modifies the underlying data source.
- This functionality – `java.util.stream` – supports functional-style operations on streams of elements, such as map-reduce transformations on collections.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.51



Example

```
private static Employee[] arrayOfEmps = {
    new Employee(1, "Jeff Bezos", 100000.0),
    new Employee(2, "Bill Gates", 200000.0),
    new Employee(3, "Mark Zuckerberg", 300000.0)};
Stream.of(arrayOfEmps);
OR
private static List<Employee> empList =
Arrays.asList(arrayOfEmps);
empList.stream();
```


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.52



Example

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;
public class StreamExample {    public static void main(String[] args) {
    List<Integer> myList = new ArrayList<>();
    for(int i=0; i<100; i++) myList.add(i);
    //sequential stream
    Stream<Integer> sequentialStream = myList.stream();
    //parallel stream
    Stream<Integer> parallelStream = myList.parallelStream();
    //using lambda with Stream API, filter example
    Stream<Integer> highNums = parallelStream.filter(p -> p > 90);
    //using lambda in forEach
    highNums.forEach(p -> System.out.println("High Nums parallel="+p));
    Stream<Integer> highNumsSeq = sequentialStream.filter(p -> p > 90);
    highNumsSeq.forEach(p -> System.out.println("High Nums sequential="+p));}}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.53




Try with Resources

A resource is an object that must be closed once your program is done using it. For example a File resource or JDBC resource for database connection or a Socket connection resource. Before Java 7, there was no auto resource management and we should explicitly close the resource once our work is done with it. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks and performance hit when we forgot to close the resource.

Java 7 try with resources implementation:


```
try{// open resources here}
    // use resources
} catch (FileNotFoundException e) {
    // exception handling}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof. BVICAM U4.54

 **Java Time API**


- It has always been hard to work with Date, Time and Time Zones in java. There was no standard approach or API in java for date and time in Java. One of the nice addition in Java 8 is the java.time package that will streamline the process of working with time in java.
- Just by looking at Java Time API packages, I can sense that it will be very easy to use. It has some sub-packages java.time.format that provides classes to print and parse dates and times and java.time.zone provides support for time-zones and their rules.
- The new Time API prefers enums over integer constants for months and days of the week. One of the useful class is DateTimeFormatter for converting datetime objects to strings.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.55

 **Collections API Improvements**

- We have already seen forEach() method and Stream API for collections. Some new methods added in Collection API are:
- Iterator default method forEachRemaining(Consumer action) to perform the given action for each remaining element until all elements have been processed or the action throws an exception.
- Collection default method removeIf(Predicate filter) to remove all of the elements of this collection that satisfy the given predicate.
- Collection spliterator() method returning Spliterator instance that can be used to traverse elements sequentially or parallel.
- Map replaceAll(), compute(), merge() methods.
- Performance Improvement for HashMap class with Key Collisions

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.56

 **Concurrency API Improvements**

- Some important concurrent API enhancements are:
- ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.
- CompletableFuture that may be explicitly completed (setting its value and status).
- Executors newWorkStealingPool() method to create a work-stealing thread pool using all available processors as its target parallelism level.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.57

Java IO Improvements

- Some IO improvements known to me are:
- Files.list(Path dir) that returns a lazily populated Stream, the elements of which are the entries in the directory.
- Files.lines(Path path) that reads all lines from a file as a Stream.
- Files.find() that returns a Stream that is lazily populated with Path by searching for files in a file tree rooted at a given starting file.
- BufferedReader.lines() that return a Stream, the elements of which are lines read from this BufferedReader.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.59

Miscellaneous Core API Improvements

- Some misc API improvements that might come handy are:


1. ThreadLocal static method withInitial(Supplier supplier) to create instance easily.
2. Comparator interface has been extended with a lot of default and static methods for natural ordering, reverse order etc.
3. min(), max() and sum() methods in Integer, Long and Double wrapper classes.
4. logicalAnd(), logicalOr() and logicalXor() methods in Boolean class.
5. ZipFile.stream() method to get an ordered Stream over the ZIP file entries. Entries appear in the Stream in the order they appear in the central directory of the ZIP file.
6. Several utility methods in Math class.
7. jjs command is added to invoke Nashorn Engine.
8. jdeps command is added to analyze class files
9. JDBC-ODBC Bridge has been removed.
10. PermGen memory space has been removed

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.59

Java 8 Memory optimization

The diagram illustrates the Java 8 memory layout. It shows a horizontal bar representing memory spaces: Eden, S0, S1, Old Memory, and Perm. Above Eden, S0, and S1, there are arrows for Minor GC. Above Old Memory and Perm, there is an arrow for Major GC. Below Eden, S0, and S1, there is a double-headed arrow for Young Gen (-Xmn). Below Old Memory and Perm, there is a double-headed arrow for JVM Heap (-Xms -Xmx). Below Perm, there are two labels: -XX:PermSize and -XX:MaxPermSize.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.60



Java 8 Memory Optimization

- The Heap memory is physically divided into two parts -
- **Young Generation and Old Generation.**
- **Memory Management in Java – Young Generation**
- The young generation is the place where all the new objects are created. When the young generation is filled, garbage collection is performed. This garbage collection is called **Minor GC**.
- Young Generation is divided into three parts - **Eden Memory** and two **Survivor Memory spaces**.
- Most of the newly created objects are located in the Eden memory space.
- When Eden space is filled with objects, Minor GC is performed and all the survivor objects are moved to one of the survivor spaces.
- Minor GC also checks the survivor objects and move them to the other survivor space. So at a time, one of the survivor space is always empty.
- Objects that are survived after many cycles of GC, are moved to the Old generation memory space. Usually, it's done by setting a threshold for the age of the young generation objects before they become eligible to promote to Old generation.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.61



Java 8 Memory Optimization

- Memory Management in Java – Old Generation
- Old Generation memory contains the objects that are long-lived and survived after many rounds of Minor GC. Usually, garbage collection is performed in Old Generation memory when it's full. Old Generation Garbage Collection is called Major GC and usually takes a longer time.
- Java Memory Model – Permanent Generation
- Permanent Generation or "Perm Gen" contains the application metadata required by the JVM to describe the classes and methods used in the application. Note that Perm Gen is not part of Java Heap memory.
- Perm Gen is populated by JVM at runtime based on the classes used by the application. Perm Gen also contains Java SE library classes and methods. Perm Gen objects are garbage collected in a full garbage collection.
- Java Memory Model – Method Area
- Method Area is part of space in the Perm Gen and used to store class structure (runtime constants and static variables) and code for methods and constructors.


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.62



Java 8 Memory Optimization


- Java Memory Model – Memory Pool
- Memory Pools are created by JVM memory managers to create a pool of immutable objects if the implementation supports it. String Pool is a good example of this kind of memory pool. Memory Pool can belong to Heap or Perm Gen, depending on the JVM memory manager implementation.
- Java Memory Model – Runtime Constant Pool
- Runtime constant pool is per-class runtime representation of constant pool in a class. It contains class runtime constants and static methods. Runtime constant pool is part of the method area.
- Java Memory Model – Java Stack Memory
- Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that is getting referred from the method.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.63

 **Java 8 Memory Optimization**

- Memory Management in Java – Java Heap Memory Switches
- Java provides a lot of memory switches that we can use to set the memory sizes and their ratios. Some of the commonly used memory switches are:
 - -XX:PermGen For setting the initial size of the Permanent Generation memory
 - -XX:MaxPermGen For setting the maximum size of Perm Gen
- Memory Management in Java – Java Garbage Collection

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.64

 **Java 8 Memory Optimization**

- One of the basic ways of garbage collection involves three steps:
 1. Marking: This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
 2. Normal Deletion: Garbage Collector removes the unused objects and re-claim the free space to be allocated to other objects.
 3. Deletion with Compacting: For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Ritika Wason, Asso. Prof, BVICAM U4.65
