

# JAVA Session 11-30

take the below session and give the short important theory that covers all the points that i want to learn and all that covers the mcq also because i want to prepare for the mcq purpose exam also  
 Session 11: Lecture: ☐ Access modifiers (public, private, protected and default) ☐ Packages and import statements ☐ Static imports ☐ Constructor chaining (with and without packages) ☐ Accessing protected variables and methods outside the package

Below is **short, exam-oriented theory** for **Session 11**, written in **simple language, point-wise**, and **MCQ focused**.

This will help you **revise fast** and **answer tricky MCQs** confidently.

## Session 11 – Important Theory (MCQ Focused)

### 1. Access Modifiers in Java

Access modifiers control **where a class, variable, method, or constructor can be accessed**.

Modifier	Same Class	Same Package	Subclass (other package)	Anywhere
<b>public</b>	✓	✓	✓	✓
<b>protected</b>	✓	✓	✓	✗
<b>default</b> (no keyword)	✓	✓	✗	✗
<b>private</b>	✓	✗	✗	✗

#### MCQ Points

- **private** members are accessible **only inside the same class**.
- **default access** is used when **no keyword is written**.
- **protected** allows access in **subclasses outside the package**.
- **public** has **no restriction**.

### 2. Packages in Java

A **package** is a **group of related classes and interfaces**.

#### Types of Packages

1. **Built-in packages** → `java.util`, `java.lang`, `java.io`
2. **User-defined packages**

## Syntax

```
java

package mypackage;
```

## Advantages

- Code organization
- Avoids class name conflicts
- Supports access protection
- Improves reusability

### MCQ Points

- A package statement **must be the first line** in a Java file.
- A class can belong to **only one package**.
- If no package is mentioned → **default package**.

## 3. import Statement

`import` is used to **access classes from another package**.

## Syntax

```
java

import java.util.Scanner;
import java.util.*;
```

### MCQ Points

- `*` imports **all classes**, not sub-packages.
- `java.lang` is **automatically imported**.
- `import` is a **compile-time instruction**, not runtime.

## 4. Static Import

`static import` allows access to **static members directly without class name**.

## Syntax

```
java

import static java.lang.Math.*;
```

## Example

```
java

System.out.println(sqrt(16)); // no Math.sqrt
```

### MCQ Points

- Used **only for static variables and methods**.
- Reduces code length.
- Introduced in **Java 5**.

## 5. Constructor Chaining

Constructor chaining means **calling one constructor from another constructor**.

### Types

1. **Within same class** → `this()`
2. **From parent class** → `super()`

### Rules

- `this()` or `super()` **must be the first statement**.
- You **cannot use both** in the same constructor.

## Example

```
java

this(); // same class
super(); // parent class
```

### MCQ Points

- If `super()` is not written, **Java inserts it automatically**.
- Constructor chaining avoids **code duplication**.
- Constructors **cannot be overridden**.

## 6. Constructor Chaining with Packages

- If parent class is in **another package**, constructor access depends on **access modifier**.

### Rules

- **public** constructor → accessible everywhere

- **protected** constructor → accessible in subclass
- **default** → same package only
- **private** → not accessible

### MCQ Points

- `super()` works only if constructor is **accessible**.
- Private constructors **cannot be called** outside the class.

## 7. Accessing protected Members Outside Package

Protected members can be accessed:

- ✓ In same package
- ✓ In subclass outside package
- ✗ Using object reference outside package

### Correct Way

```
java

class Child extends Parent {
    void show() {
        System.out.println(x); // allowed
    }
}
```

### Wrong Way

```
java

Parent p = new Parent();
p.x; // NOT allowed outside package
```

### MCQ Points

- Protected members are accessed via **inheritance**, not object.
- Subclass reference is required.

## Most Important MCQ Traps

- **protected** ≠ **public**
- **default** means **no keyword**
- `java.lang` needs **no import**
- **static import** works only for **static members**
- `this()` and `super()` **cannot be together**

- Protected members **cannot be accessed using object reference outside package**

If you want, I can:

- ✓ Give **30–50 MCQs** only from **Session 11**
- ✓ Give **previous exam-type MCQs**
- ✓ Create **1-page revision notes PDF**
- ✓ Combine this with **Session 1–10 full MCQ revision**

Just tell me 👍

Session 12:

Lecture: ☐ Garbage collection in java

☐ Requesting JVM to run garbage collection. ☐ Different ways to make object eligible for garbage collection: (Nulling a reference variable, Re- assigning a reference variable & island of isolation) ☐ Finalize method.

Below is **short, crisp, MCQ-oriented theory** for **Session 12**, written in **easy language** and covering **all exam-important points**.

## Session 12 – Garbage Collection in Java (MCQ Focused Notes)

### 1. Garbage Collection (GC) in Java

**Garbage Collection** is the **automatic process** by which JVM **removes unused objects from heap memory**.

#### Key Points

- GC is handled by **JVM**, not the programmer.
- GC works on **heap memory only**.
- An object is eligible for GC when it has **no reference**.

#### 🔑 MCQ Points

- Java does **not support manual memory deallocation**.
- GC **frees memory**, not variables.
- GC runs in a **separate low-priority thread**.

### 2. Requesting JVM to Run Garbage Collection

Java allows us to **request**, but **not force**, GC.

## Methods

```
java

System.gc();
Runtime.getRuntime().gc();
```

## Important Notes

- These methods **only request** GC.
- JVM **may or may not run** GC.

## MCQ Points

- `System.gc()` does **not guarantee** GC.
- GC decision is made by **JVM**, not program.

# 3. Object Eligible for Garbage Collection

An object becomes eligible when **no active reference points to it**.

## A. Nulling a Reference Variable

```
java

Student s = new Student();
s = null;
```

✓ Object becomes eligible for GC.

## MCQ Point

- Setting reference to `null` **does not destroy object immediately**.

## B. Reassigning a Reference Variable

```
java

Student s = new Student();
s = new Student();
```

✓ First object becomes eligible for GC.

## MCQ Point

- Old object loses reference → eligible for GC.

## C. Island of Isolation

Objects **reference each other** but **no external reference exists**.

```
java

A a = new A();
B b = new B();
a.b = b;
b.a = a;

a = null;
b = null;
```

✓ Both objects are eligible for GC.

### MCQ Points

- GC **can handle circular references**.
- Reference counting **fails**, but JVM GC works.

## 4. Finalize Method

`finalize()` is called by GC **before destroying an object**.

### Syntax

```
java

protected void finalize() throws Throwable {
    // cleanup code
}
```

### Key Points

- Called **only once** per object.
- Not guaranteed to be called.
- Deprecated in **Java 9+**.

### MCQ Points

- `finalize()` is **not reliable**.
- Programmer should **not depend** on it.
- It is **not a destructor**.

## Important Differences (MCQ Favorite)

Concept	Explanation
<b>final</b>	Keyword (constant)
<b>finally</b>	Block (exception handling)
<b>finalize()</b>	Method (GC related)

## Most Common MCQ Traps


- `System.gc()`  $\neq$  force GC
- GC destroys **objects**, not references
- Circular references **do not prevent GC**
- `finalize()` may **never be executed**
- GC runs on **heap memory only**
- Java has **no destructor**

## One-Line Revision (Last-Minute)

- GC is **automatic**
- JVM decides **when GC runs**
- Objects with **no reference**  $\rightarrow$  GC
- `null`, reassignment, island  $\rightarrow$  GC eligible
- `finalize()` is **unreliable**

If you want next:

- ✓ **25-40 MCQs** from Session 12
- ✓ **Session 11 + 12 combined MCQs**
- ✓ **Exam-pattern tricky MCQs with answers**
- ✓ **1-page ultra-short revision notes**

Just tell me 

Sessions 13 & 14:

Wrapper Classes and String Class

Lecture: ☐ Wrapper classes and constant pools ☐ String class, StringBuffer & StringBuilder class ☐ String pool

Below are **short, high-yield, MCQ-oriented notes** for **Sessions 13 & 14**, written in **easy language** and covering **all common exam traps**.



# Sessions 13 & 14 – Wrapper Classes & String (MCQ Focused Notes)

## 1. Wrapper Classes in Java

Wrapper classes convert **primitive data types into objects**.

Primitive	Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

### Why Wrapper Classes?

- Needed for **Collections** (they store objects).
- Provide **utility methods**.
- Support **object representation of primitives**.

### MCQ Points

- Wrapper classes are in **java.lang**.
- Wrapper classes are **immutable**.
- Each primitive has **one wrapper class**.

## 2. Constant Pool (Wrapper Constant Pool)

Java stores commonly used wrapper objects in **constant pool** for memory efficiency.

### Range Stored

- **Integer** → **-128 to 127**
- Same for Byte, Short, Long
- **Character** → **\u0000 to \u007F**
- **Boolean** → **true and false**

### Example

```
java

Integer a = 100;
Integer b = 100;    // same object
Integer c = 200;
Integer d = 200;    // different objects
```

### MCQ Points

- Constant pool works with **autoboxing**.
- Outside range → **new object created**.
- == may return true **inside pool range**.

## 3. Autoboxing and Unboxing

- **Autoboxing** → primitive → wrapper
- **Unboxing** → wrapper → primitive

```
java

Integer i = 10;    // autoboxing
int x = i;         // unboxing
```

### MCQ Points

- Introduced in **Java 5**
- Happens **automatically**

## 4. String Class

A **String** is a sequence of characters.

### Key Properties

- Strings are **immutable**
- Stored in **String Constant Pool (SCP)**

### Example

```
java

String s1 = "Java";
String s2 = "Java";    // same object
```

### MCQ Points

- Modifying a String creates **new object**

- String class is **final**
- Stored in **heap + SCP**

## 5. String Constant Pool (String Pool)

String Pool stores **unique string literals**.

```
java

String s1 = "Hello";
String s2 = "Hello";
```

✓ Both point to **same object**

### Using new Keyword

```
java

String s3 = new String("Hello");
```

✓ Creates **two objects** (heap + pool if not present)

### MCQ Points

- Pool avoids **duplicate objects**
- **intern()** returns reference from pool
- SCP is part of **heap memory**

## 6. StringBuffer Class

- **Mutable**
- **Thread-safe**
- Slower than StringBuilder

```
java

StringBuffer sb = new StringBuffer("Hi");
sb.append(" Java");
```

### MCQ Points

- All methods are **synchronized**
- Introduced in **Java 1.0**

## 7. StringBuilder Class

- **Mutable**
- **Not thread-safe**
- Faster than StringBuffer

```
java

StringBuilder sb = new StringBuilder("Hi");
sb.append(" Java");
```

## MCQ Points

- Introduced in **Java 5**
- Used in **single-threaded** environment

## 8. String vs StringBuffer vs StringBuilder

Feature	String	StringBuffer	StringBuilder
Mutability	✗ Immutable	✓ Mutable	✓ Mutable
Thread-safe	✓	✓	✗
Performance	Slow	Medium	Fast
Memory	More	Less	Less

## 9. == vs equals() (Very Important MCQs)

- == → compares **reference**
- equals() → compares **content**

```
java

String s1 = "Java";
String s2 = new String("Java");

s1 == s2      // false
s1.equals(s2) // true
```

## Most Common MCQ Traps

- Wrapper classes are **immutable**
- Integer pool range is **-128 to 127**
- String is **final & immutable**
- **new String()** creates **extra object**

- StringBuilder is **not thread-safe**
- == checks reference, not value

## One-Line Last-Minute Revision

- Wrapper = object version of primitive
- Constant pool saves memory
- String → immutable
- StringBuffer → thread-safe
- StringBuilder → fastest
- Pool objects may give == `true`

If you want next:

- ✓ **40-60 MCQs** from Sessions 13 & 14
- ✓ **Tricky pool-based MCQs**
- ✓ **One-page ultra-revision sheet**
- ✓ **Session 11-14 combined MCQs**

Just tell me 👍

Sessions 15 & 16:

Exception Handling

Lecture: □ Exception hierarchy, Errors, Checked and un-checked exceptions. □ Exception propagation

□ try-catch-finally block, throws clause and throw keyword. □ Multi catch block. □ Creating user defined checked and unchecked exceptions.

Below are **short, exam-oriented, MCQ-focused notes** for **Sessions 15 & 16 – Exception Handling**, written in **simple language** and covering **all important traps**.

## Sessions 15 & 16 – Exception Handling (MCQ Focused Notes)

### 1. Exception Handling in Java

**Exception** = an **abnormal situation** that occurs at **runtime** and disrupts normal program flow.

#### Purpose

- Prevent program crash
- Maintain normal flow

- Provide meaningful error message

## 2. Exception Hierarchy (Very Important MCQs)



## 3. Errors vs Exceptions

### Errors

- Caused by **JVM**
- Cannot be handled by program
- Examples:
  - `OutOfMemoryError`
  - `StackOverflowError`

### Exceptions

- Caused by **program logic**
- Can be handled using try-catch

### MCQ Points

- Errors are **unchecked**
- Programs should **not handle Errors**

## 4. Checked vs Unchecked Exceptions

### Checked Exceptions

- Checked at **compile time**
- Must be **handled or declared**
- Examples:
  - `IOException`

- `SQLException`
- `ClassNotFoundException`

## Unchecked Exceptions

- Checked at **runtime**
- No need to handle compulsorily
- Examples:
  - `NullPointerException`
  - `ArithmeticException`
  - `ArrayIndexOutOfBoundsException`

### MCQ Points

- Checked → compiler forces handling
- Unchecked → subclasses of `RuntimeException`

## 5. Exception Propagation

- When an exception occurs in a method:
  - If not handled → passed to calling method
- This continues until:
  - Exception is handled
  - Or program terminates

### MCQ Points

- Applies only to **unchecked exceptions**
- Checked exceptions need **throws**

## 6. try-catch-finally Block

### Syntax

```
java

try {
    // risky code
} catch(Exception e) {
    // handling code
} finally {
    // cleanup code
}
```

### finally Block

- Always executes
- Used for **closing resources**

### MCQ Points

- finally executes even if exception is not raised
- finally does NOT execute if `System.exit()` is called
- Multiple catch blocks allowed

## 7. throws Clause

Used to **declare exceptions** to caller.

```
java

void readFile() throws IOException {
}
```

### MCQ Points

- Used with **method declaration**
- Mainly for **checked exceptions**
- Can throw multiple exceptions

## 8. throw Keyword

Used to **explicitly create and throw an exception**.

```
java

throw new ArithmeticException("Error");
```

### MCQ Points

- Used inside method body
- Can throw **only one exception at a time**
- Exception object must be created

## 9. throws vs throw (MCQ Favorite)

throw	throws
Used to create exception	Used to declare exception
Inside method body	In method signature



throw	throws
Throws one exception	Can declare multiple
Followed by object	Followed by class name

## 10. Multi-Catch Block (Java 7)

Handles multiple exceptions in **one catch**.

```
java

catch(IOException | SQLException e) {
}
```

### Rules

- Exceptions must be **unrelated**
- Parent and child together → **compile error**

### MCQ Points

- Introduced in **Java 7**
- Reduces code duplication

## 11. User-Defined Exceptions

### A. Checked Exception

Extend **Exception**

```
java

class MyException extends Exception {
    MyException(String msg) {
        super(msg);
    }
}
```

### B. Unchecked Exception

Extend **RuntimeException**

```
java

class MyException extends RuntimeException {
    MyException(String msg) {
        super(msg);
    }
}
```

## MCQ Points

- Checked → must be handled
- Unchecked → optional handling
- Custom exceptions improve readability

## Most Common MCQ Traps


- **Throwable** is parent of all exceptions
- Errors are **not recoverable**
- **finally** always runs except **System.exit()**
- Only **unchecked** exceptions propagate automatically
- Multi-catch cannot contain parent-child
- **throw** ≠ **throws**

## One-Line Last-Minute Revision

- Exception → runtime problem
- Checked → compile-time
- Unchecked → runtime
- **throw** creates exception
- **throws** declares exception
- **finally** cleans resources

If you want next:

- ✓ **50-70 MCQs** from Sessions 15 & 16
- ✓ **Exception-based tricky MCQs**
- ✓ **Session 11-16 combined MCQs**
- ✓ **Quick revision chart PDF**

Just tell me 

Sessions 17:

java.io & java.nio Package

Lecture: □ Brief introduction to InputStream, OutputStream, Reader and Writer interfaces □ NIO package

□ Serialization and de-serialization

□ Shallow copy and deep copy

Below are **short, high-yield, MCQ-oriented notes** for **Session 17 – java.io & java.nio**, written in **easy language** and covering **all exam-important traps**.

## Session 17 – java.io & java.nio (MCQ Focused Notes)

### 1. java.io Package (Overview)

java.io provides **classes and interfaces for input and output operations** (file, console, network).

#### Two Types of Streams

1. **Byte Streams**
2. **Character Streams**

### 2. InputStream & OutputStream (Byte Streams)

Used for **binary data** (images, audio, video).

#### Key Interfaces

- **InputStream** → read data
- **OutputStream** → write data

#### Common Classes

- **FileInputStream**
- **FileOutputStream**
- **BufferedInputStream**
- **BufferedOutputStream**

#### MCQ Points

- Byte streams handle **8-bit data**
- Suitable for **non-text files**
- Parent class → **InputStream**

### 3. Reader & Writer (Character Streams)

Used for **text data**.

#### Key Interfaces

- **Reader** → read characters

- **Writer** → write characters

## Common Classes

- **FileReader**
- **FileWriter**
- **BufferedReader**
- **BufferedWriter**

## MCQ Points

- Character streams handle **16-bit Unicode**
- Suitable for **text files**
- More efficient for text than byte streams

## 4. InputStream vs Reader (MCQ Favorite)

Feature	InputStream	Reader
Data Type	Byte	Character
Best For	Binary files	Text files
Encoding	No	Yes
Size	8-bit	16-bit

## 5. java.nio Package (New I/O)

Introduced in **Java 1.4** to improve performance.

### Key Features

- **Buffer-based** data handling
- **Channel-based** I/O
- **Non-blocking** I/O
- Faster than **java.io**

### Core Components

- **Buffer**
- **Channel**
- **Selector**

## MCQ Points

- NIO supports **non-blocking I/O**
- Used for **high-performance applications**
- Path & Files classes introduced in **Java 7**

## 6. Serialization

Serialization is the process of **converting an object into byte stream** for storage or transfer.

### How?

- Class must implement **Serializable**
- Uses **ObjectOutputStream**

```
java

class Student implements Serializable {
}
```

### MCQ Points

- Marker interface (no methods)
- **transient** variables are **not serialized**
- Static variables are **not serialized**

## 7. De-serialization

Process of **reconstructing object from byte stream**.

- Uses **ObjectInputStream**

### MCQ Points

- Constructor is **not called**
- Same class version required
- Uses **serialVersionUID**

## 8. Serialization Important Terms

Term	Meaning
<b>Serializable</b>	Marker interface
<b>transient</b>	Skip during serialization
<b>static</b>	Not serialized
<b>serialVersionUID</b>	Version control

## 9. Shallow Copy

- Copies **object reference**
- Changes affect **both objects**

```
java
Object obj2 = obj1;
```

### MCQ Points

- Shared references
- Faster
- Risky

## 10. Deep Copy

- Copies **actual object**
- Independent objects

```
java
Object obj2 = new Object(obj1);
```

### MCQ Points

- Separate memory
- Safe
- Slower

## 11. Shallow vs Deep Copy (Very Important)

Feature	Shallow Copy	Deep Copy
Reference	Shared	Separate
Memory	Less	More
Changes	Reflect	No effect
Speed	Fast	Slow

## Most Common MCQ Traps

- Byte streams  $\neq$  character streams
- NIO  $\neq$  blocking always
- **Serializable** has **no methods**
- Constructor not called in deserialization
- **transient** fields ignored
- Shallow copy shares references

## One-Line Last-Minute Revision

- **InputStream**  $\rightarrow$  bytes
- **Reader**  $\rightarrow$  characters
- NIO  $\rightarrow$  fast & non-blocking
- Serialization  $\rightarrow$  object  $\rightarrow$  byte
- Deserialization  $\rightarrow$  byte  $\rightarrow$  object
- Deep copy  $\rightarrow$  safe copy

If you want next:

- ✓ **40-60 MCQs** from Session 17
- ✓ **java.io vs java.nio tricky MCQs**
- ✓ **Serialization-based MCQs**
- ✓ **Sessions 11-17 full MCQ set**

Just tell me 

Session 18:

Lecture:

Object Class & java.util Package

□ Date, DateTime, Calendar class □ Converting Date to String and String to Date using SimpleDateFormat class □ Object Class: Overriding toString, equals & hashCode method

Below are **short, exam-oriented, MCQ-focused notes** for **Session 18 – Object Class & java.util Package**, written in **simple language** and covering **all high-probability MCQ traps**.

## Session 18 – Object Class & java.util (MCQ Focused Notes)

### 1. java.util Package (Overview)

`java.util` contains **utility classes** such as:

- Date & time classes
- Collections
- Formatter & Scanner
- Random, Locale, etc.

### MCQ Point

- `java.util` is one of the **most used packages** in Java.

## 2. Date Class (`java.util.Date`)

Represents **date and time** (milliseconds since **1 Jan 1970 UTC**).

```
java
Date d = new Date();
```

### Key Points

- Stores both **date + time**
- Many methods are **deprecated**
- Mostly replaced by `java.time` API

### MCQ Points

- Date stores time in **milliseconds**
- Date is **mutable**
- Exists since **Java 1.0**

## 3. Calendar Class (`java.util.Calendar`)

An **abstract class** used for advanced date operations.

```
java
Calendar c = Calendar.getInstance();
```

### Features

- Can modify date fields
- Supports different locales & time zones

### MCQ Points



- Calendar is **abstract**
- Month is **0-based** (Jan = 0)
- Used before Java 8

## 4. DateTime API (Java 8+)

Located in `java.time` package.

### Common Classes

- `LocalDate` → date only
- `LocalTime` → time only
- `LocalDateTime` → date + time

### MCQ Points

- DateTime classes are **immutable**
- Thread-safe
- Preferred over Date & Calendar

## 5. SimpleDateFormat Class

Used to **convert Date ↔ String**.

### Syntax

```
java

SimpleDateFormat sdf =
    new SimpleDateFormat("dd-MM-yyyy");
```

### A. Date → String

```
java

String s = sdf.format(new Date());
```

### B. String → Date

```
java

Date d = sdf.parse("24-12-2025");
```

### MCQ Points

- `format()` → Date → String
- `parse()` → String → Date
- Not thread-safe
- Part of `java.text` package

## 6. Object Class (Very Important)

`Object` is the **parent class of all Java classes**.

### Key Methods

- `toString()`
- `equals()`
- `hashCode()`

### MCQ Points

- Every class **implicitly extends Object**
- Object class is in **`java.lang`**

## 7. Overriding `toString()`

Used to provide **meaningful string representation** of object.

### Default Behavior

```
java
ClassName@HexHashCode
```

### MCQ Points

- Automatically called when object is printed
- Overriding improves readability

## 8. Overriding `equals()`

Used to compare **object content**, not reference.

### Default Behavior

- Same as `==`

### MCQ Points

- Used in collections
- Must override with `hashCode()`

## 9. Overriding `hashCode()`

Returns **unique integer value** for object.

### Rules (MCQ Favorite)

- Equal objects → same `hashCode`
- Same `hashCode` ≠ equal objects

### MCQ Points

- Used by `HashMap`, `HashSet`
- Contract with `equals()`

## 10. `equals()` vs `==` (Very Important)

Feature	<code>==</code>	<code>equals()</code>
Compares	Reference	Content
Used for	Objects & primitives	Objects only
Can be overridden	✗	✓

## 11. `equals()` & `hashCode()` Contract

- If `equals()` is overridden → `hashCode()` must be overridden
- Violating contract causes **collection issues**

## Most Common MCQ Traps

- Calendar month starts from **0**
- Date is **mutable**, `DateTime` is **immutable**
- `SimpleDateFormat` is **not thread-safe**
- `Object` is parent of all classes
- `equals` without `hashCode` → **bug**
- `toString()` is auto-called

# One-Line Last-Minute Revision

- Date → old API
- Calendar → flexible but old
- DateTime → Java 8+, best
- SimpleDateFormat → parse & format
- Object → parent of all
- equals + hashCode → always together

If you want next:

- ✓ **40-60 MCQs** from Session 18
- ✓ **Date & Object-based tricky MCQs**
- ✓ **Sessions 11-18 full MCQ revision**
- ✓ **One-page ultra-revision sheet**

Just tell me 👍

Sessions 19, 20, 21 & 22:

Lecture: □ Introduction to collections: Collection hierarchy □ List, Queue, Set and Map Collections □

List Collection:

- o ArrayList, LinkedList
- o Vector (insert, delete, search, sort, iterate, replace operations) □ Collections class □ Comparable and Comparator interfaces □ Queue collection

Below are **short, high-yield, MCQ-oriented notes** for **Sessions 19-22 (Collections Framework)**, written in **easy language, point-wise**, and covering **all exam traps**.

## Sessions 19-22 – Collections Framework (MCQ Focused Notes)

### 1. Introduction to Collections

**Collection Framework** is a **set of classes and interfaces** used to **store and manipulate groups of objects**.

#### Advantages

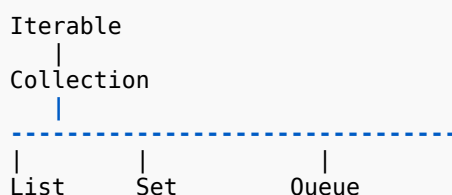
- Dynamic size
- Ready-made data structures
- Better performance & reusability

#### MCQ Points

- Collections store **objects only**
- Introduced in **Java 1.2**
- Located in **java.util**

## 2. Collection Hierarchy (Very Important MCQs)

markdown



javascript

Map (separate hierarchy)

### MCQ Points

- Map does **not extend Collection**
- Iterable allows **for-each loop**

## 3. List Interface

- Allows **duplicate elements**
- Maintains **insertion order**
- Allows **index-based access**

### Common Classes

- ArrayList
- LinkedList
- Vector

## 4. ArrayList

- Uses **dynamic array**
- Fast **random access**
- Slow insertion/deletion (middle)

### MCQ Points

- Not synchronized
- Allows duplicates & nulls
- Best for **search operations**

## 5. LinkedList

- Uses **doubly linked list**
- Fast insertion/deletion
- Slow random access

### MCQ Points

- Implements **List & Deque**
- Can act as **Queue**
- Allows duplicates & nulls

## 6. Vector

- Legacy class
- **Thread-safe (synchronized)**
- Slower than ArrayList

### MCQ Points

- Introduced in **Java 1.0**
- Methods are synchronized
- Enumeration used instead of Iterator (legacy)

## Common Vector Operations

Operation	Method
Insert	<b>add()</b>
Delete	<b>remove()</b>
Search	<b>contains()</b>
Sort	<b>Collections.sort()</b>
Iterate	<b>Iterator, Enumeration</b>
Replace	<b>set(index, value)</b>

## 7. List vs ArrayList vs LinkedList (MCQ Favorite)

Feature	ArrayList	LinkedList
Structure	Array	Linked List
Search	Fast	Slow
Insert/Delete	Slow	Fast
Memory	Less	More

## 8. Set Interface

- **No duplicates**
- No guaranteed order (except some)

### Common Classes

- HashSet
- LinkedHashSet
- TreeSet

### MCQ Points

- HashSet allows **one null**
- TreeSet does **not allow null**
- Set uses `equals()` & `hashCode()`

## 9. Map Interface

- Stores **key-value pairs**
- Keys must be **unique**

### Common Classes

- HashMap
- LinkedHashMap
- TreeMap

### MCQ Points

- Map is **not a Collection**
- HashMap allows **one null key**

- TreeMap sorts by **key**

## 10. Collections Class

Utility class to perform operations on collections.

### Common Methods

```
java

Collections.sort(list);
Collections.reverse(list);
Collections.shuffle(list);
Collections.max(list);
```

### MCQ Points

- Part of **java.util**
- Contains **static methods only**
- Differs from **Collection** interface

## 11. Comparable Interface

Used for **natural sorting**.

```
java

class Student implements Comparable<Student> {
    public int compareTo(Student s) {
        return this.id - s.id;
    }
}
```

### MCQ Points

- Method → **compareTo()**
- Only **one sorting logic**
- Part of **java.lang**

## 12. Comparator Interface

Used for **custom sorting**.

```
java

class AgeComparator implements Comparator<Student> {
    public int compare(Student a, Student b) {
        return a.age - b.age;
    }
}
```



```
}  
}
```

### MCQ Points

- Method → `compare()`
- Multiple sorting logics possible
- Part of `java.util`

## 13. Comparable vs Comparator (Very Important)

Feature	Comparable	Comparator
Package	<code>java.lang</code>	<code>java.util</code>
Method	<code>compareTo()</code>	<code>compare()</code>
Sorting	Natural	Custom
Modification	Class needed	No class change

## 14. Queue Interface

- Follows **FIFO**
- Used for **processing tasks**

### Common Classes

- `PriorityQueue`
- `ArrayDeque`
- `LinkedList`

### Common Methods

Method	Meaning
<code>add()</code>	insert
<code>poll()</code>	remove
<code>peek()</code>	view head

### MCQ Points

- `PriorityQueue` does **not follow insertion order**
- Nulls **not allowed** in `PriorityQueue`

- Queue extends Collection

## Most Common MCQ Traps


- Map  $\neq$  Collection
- Vector is synchronized
- TreeSet does not allow null
- Comparable  $\rightarrow$  one logic
- Comparator  $\rightarrow$  multiple logics
- PriorityQueue  $\neq$  FIFO always

## One-Line Last-Minute Revision

- List  $\rightarrow$  duplicates allowed
- Set  $\rightarrow$  no duplicates
- Map  $\rightarrow$  key-value
- ArrayList  $\rightarrow$  fast search
- LinkedList  $\rightarrow$  fast insert
- Comparable  $\rightarrow$  natural
- Comparator  $\rightarrow$  custom

If you want next:

- ✓ **70-100 MCQs** from Sessions 19-22
- ✓ **Tricky Comparable/Comparator MCQs**
- ✓ **Collections interview + exam combo MCQs**
- ✓ **Sessions 1-22 full MCQ revision**

Just tell me 

Sessions 23, 24, 25 & 26:

Lecture: ☐ Set Collection:

o HashSet, LinkedHashSet&TreeSet collection

o Backed set collections. ☐ Map Collection:

o HashTable, HashMap, LinkedHashMap&TreeMap classes

o Backed Map collections. ☐ Concurrent collections ☐ Implementation of Java 8 stream API

Below are **short, exam-oriented, MCQ-focused notes** for **Sessions 23-26**, written in **simple language, point-wise**, and covering **all high-probability traps**.

# Sessions 23–26 – Set, Map, Concurrent Collections & Java 8 Stream API (MCQ Focused Notes)

---

## 1. Set Collection (Overview)

**Set** stores **unique elements** (no duplicates).

### Key Points

- Does **not allow duplicates**
  - Uses `equals()` and `hashCode()`
  - Part of `java.util`
- 

## 2. HashSet

- No insertion order
- Uses **hashing**
- Fast performance

### MCQ Points

- Allows **one null**
  - Not synchronized
  - Order is **not predictable**
- 

## 3. LinkedHashSet

- Maintains **insertion order**
- Slower than HashSet

### MCQ Points

- Allows **one null**
  - Uses **hash table + linked list**
- 

## 4. TreeSet

- Stores elements in **sorted order**
- Uses **Red-Black Tree**

### MCQ Points

- Does **not** allow null
- Elements must be **Comparable**
- Sorting is **natural order**

## 5. HashSet vs LinkedHashSet vs TreeSet (Very Important)

Feature	HashSet	LinkedHashSet	TreeSet
Order	No	Insertion	Sorted
Null	One	One	✗
Speed	Fast	Medium	Slow
Structure	Hash Table	Hash + List	Tree

## 6. Backed Set Collections

Backed collections are **views backed by original collection**.

### Example

```
java
Set<Integer> s = map.keySet();
```

### MCQ Points

- Changes in one reflect in other
- Not independent copy
- Fail-fast behavior

## 7. Map Collection (Overview)

Stores **key-value pairs**.

### Key Rules

- Keys must be **unique**
- Values can be duplicated

## 8. Hashtable

- **Legacy class**

- **Synchronized**
- Slower

### MCQ Points

- No null key or value
  - Thread-safe
  - Introduced in **Java 1.0**
- 

## 9. HashMap

- Most commonly used Map
- Not synchronized

### MCQ Points

- Allows **one null key**, multiple null values
  - Faster than Hashtable
  - Introduced in **Java 1.2**
- 

## 10. LinkedHashMap

- Maintains **insertion order**
- Slightly slower than HashMap

### MCQ Points

- Allows one null key
  - Used for **LRU cache**
- 

## 11. TreeMap

- Stores keys in **sorted order**
- Uses **Red-Black Tree**

### MCQ Points

- Does **not allow null key**
  - Sorting based on **key**
  - Slower than HashMap
-

## 12. HashMap vs Hashtable (MCQ Favorite)

Feature	HashMap	Hashtable
Thread-safe	✗	✓
Null key	One	✗
Performance	Fast	Slow
Legacy	✗	✓

## 13. Backed Map Collections

Views backed by Map:

Method	Returns
<code>keySet()</code>	Set of keys
<code>values()</code>	Collection of values
<code>entrySet()</code>	Set of entries

### MCQ Points

- Changes reflect in Map
- Not independent copy

## 14. Concurrent Collections

Used in **multi-threaded environment**.

### Common Classes

- `ConcurrentHashMap`
- `CopyOnWriteArrayList`
- `CopyOnWriteArraySet`

### MCQ Points

- Better than synchronized collections
- No `ConcurrentModificationException`
- High performance

## 15. ConcurrentHashMap

- Thread-safe
- Allows concurrent read & write

### MCQ Points

- No null key or value
  - Faster than Hashtable
  - Uses **segment locking**
- 

## 16. Java 8 Stream API

Used to process collections in **functional style**.

### Key Features

- Supports **lambda expressions**
  - Improves readability
  - Supports parallel processing
- 

## 17. Stream Operations

### Intermediate Operations

- `filter()`
- `map()`
- `sorted()`

### Terminal Operations

- `forEach()`
- `collect()`
- `count()`
- `reduce()`

### MCQ Points

- Streams do **not store data**
  - Lazy evaluation
  - Stream can be used **only once**
- 

## 18. Example Stream Code (Conceptual)

```
java

list.stream()
    .filter(x -> x > 10)
    .map(x -> x * 2)
    .forEach(System.out::println);
```

## 19. Stream vs Collection (Important)

Feature	Stream	Collection
Storage	✗	✓
Reusable	✗	✓
Lazy	✓	✗
Functional	✓	✗

## Most Common MCQ Traps

- TreeSet & TreeMap do not allow null
- HashMap allows one null key
- ConcurrentHashMap does not allow null
- Streams are one-time use
- Backed collections reflect changes
- Hashtable is legacy & synchronized

## One-Line Last-Minute Revision

- Set → unique
- HashSet → fast
- TreeSet → sorted
- HashMap → one null key
- ConcurrentHashMap → thread-safe & fast
- Stream → functional & lazy

If you want next:

- ✓ **80-120 MCQs** from Sessions 23-26
- ✓ **Stream API tricky MCQs**
- ✓ **Concurrent collection comparison MCQs**
- ✓ **Sessions 1-26 full MCQ revision set**



Just tell me 👍

Session 27:

Lecture: ☐ MultiThreading : Thread class and Runnable Interface

☐ sleep, join, yield, setPriority, getPriority methods. ☐ ThreadGroup class

Below are **short, exam-oriented, MCQ-focused notes** for **Session 27 – Multithreading**, written in **simple language, point-wise**, and covering **all common exam traps**.

## Session 27 – Multithreading (MCQ Focused Notes)

### 1. Multithreading in Java

**Multithreading** allows a program to **execute multiple threads simultaneously**.

#### Advantages

- Better CPU utilization
- Faster execution
- Improved responsiveness

#### 🔑 MCQ Points

- Java supports multithreading at **language level**
- Threads share **same memory space**

### 2. Thread Class

A thread can be created by **extending Thread class**.

```
java

class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running");
    }
}
```

#### Start Thread

```
java

MyThread t = new MyThread();
t.start();    // creates new thread
```

## 🔑 MCQ Points

- `start()` creates a **new thread**
- Calling `run()` directly → **no new thread**
- Thread class is in **java.lang**

## 3. Runnable Interface

A thread can be created by **implementing Runnable**.

```
java

class MyTask implements Runnable {
    public void run() {
    }
}

Thread t = new Thread(new MyTask());
t.start();
```

## 🔑 MCQ Points

- Functional interface (one method)
- Supports **multiple inheritance**
- Preferred approach

## 4. Thread vs Runnable (MCQ Favorite)

Feature	Thread	Runnable
Inheritance	No other class	Supports inheritance
Memory	More	Less
Flexibility	Less	More
Recommended	✗	✓

## 5. sleep() Method

Puts current thread to **sleep for given time**.

```
java

Thread.sleep(1000);
```

## 🔑 MCQ Points

- Static method
- Throws **InterruptedException**
- Does **not release lock**

## 6. join() Method

Makes current thread **wait for another thread to finish**.

```
java
t.join();
```

### MCQ Points

- Non-static method
- Throws **InterruptedException**
- Used for **thread coordination**

## 7. yield() Method

Temporarily **pauses current thread** to give chance to others.

```
java
Thread.yield();
```

### MCQ Points

- Static method
- Hint to scheduler
- No guarantee

## 8. setPriority() & getPriority()

Controls thread execution priority.

```
java
t.setPriority(Thread.MAX_PRIORITY);
```

### Priority Range

- MIN\_PRIORITY → 1
- NORM\_PRIORITY → 5

- MAX\_PRIORITY → 10

### MCQ Points

- Priority is **suggestion**
- Default priority → 5
- Higher priority  $\neq$  guaranteed execution

## 9. ThreadGroup Class

Groups multiple threads into **one unit**.

```
java
ThreadGroup tg = new ThreadGroup("MyGroup");
```

### Uses

- Manage multiple threads
- Set common priority
- Interrupt all threads together

### MCQ Points

- ThreadGroup is in **java.lang**
- Parent group exists
- Mostly legacy concept

## 10. Thread Life Cycle (MCQ Important)

```
sql
New → Runnable → Running → Waiting/Blocked → Terminated
```

### MCQ Points

- **start()** → Runnable
- **sleep()** → Waiting
- **run()** completion → Dead

## Most Common MCQ Traps

- **start()**  $\neq$  **run()**

- sleep() does not release lock
- yield() is only a hint
- Runnable is preferred
- Thread priority is not guarantee
- ThreadGroup is rarely used

## One-Line Last-Minute Revision

- Thread → class
- Runnable → interface
- sleep → pause
- join → wait
- yield → give chance
- Priority → 1 to 10
- ThreadGroup → group threads

If you want next:

- ✓ 50-70 MCQs from Session 27
- ✓ Tricky multithreading MCQs
- ✓ Sessions 1-27 full MCQ revision
- ✓ Thread life-cycle diagram MCQs

Just tell me 👍

Sessions 28 & 29:

Lecture: ☐ Synchronization

☐ Deadlock ☐ Wait, notify and notifyAll methods. ☐ Producer & Consumer problem

Below are **short, exam-oriented, MCQ-focused notes** for **Sessions 28 & 29 – Synchronization & Inter-Thread Communication**, written in **simple language, point-wise**, and covering **all common exam traps**.

## Sessions 28 & 29 – Synchronization, Deadlock & Inter-Thread Communication (MCQ Focused Notes)

### 1. Synchronization in Java

**Synchronization** is used to **control access to shared resources** by multiple threads.

## Why Needed?

- Prevent **race condition**
- Ensure **data consistency**

## How to Achieve Synchronization

### 1. Synchronized method

### 2. Synchronized block

```
java

synchronized void display() { }
```

```
java

synchronized(obj) { }
```

### MCQ Points

- Synchronization works on **object lock (monitor)**
- Only **one thread** can hold lock at a time
- Reduces **performance**

## 2. Types of Synchronization

### A. Object-level Synchronization

- Lock is on **object**
- Different objects → different locks

### B. Class-level Synchronization

- Lock is on **Class object**
- Uses **static synchronized**

### MCQ Points

- Static synchronized → class-level lock
- Non-static synchronized → object-level lock

## 3. Deadlock

**Deadlock** occurs when **two or more threads wait forever** for each other's locks.

## Conditions for Deadlock (MCQ Favorite)

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

### MCQ Points

- JVM **cannot detect or resolve deadlock**
  - Program hangs indefinitely
- 

## 4. wait(), notify(), notifyAll()

Used for **inter-thread communication**.

### Key Rules

- Must be called inside **synchronized block**
  - Defined in **Object class**, not Thread
- 

### wait()

- Releases lock
- Thread goes to **waiting state**

```
java  
  
obj.wait();
```

---

### notify()

- Wakes **one waiting thread**

```
java  
  
obj.notify();
```

---

### notifyAll()

- Wakes **all waiting threads**

```
java
obj.notifyAll();
```

### MCQ Points

- wait() releases lock, sleep() does not
- notify does not release lock immediately
- These methods belong to **Object class**

## 5. wait() vs sleep() (Very Important)

Feature	wait()	sleep()
Package	Object	Thread
Lock release	✓ Yes	✗ No
Sync needed	✓ Yes	✗ No
Purpose	Communication	Pause

## 6. Producer-Consumer Problem

Classic example of **inter-thread communication**.

### Producer

- Produces data
- Calls **notify()**

### Consumer

- Consumes data
- Calls **wait()**

### MCQ Points

- Uses shared buffer
- Prevents over-production & under-consumption
- Uses **wait()** & **notify()**

## 7. How Producer-Consumer Works



1. Producer checks if buffer is full
  2. If full → wait()
  3. Consumer consumes → notify()
  4. Producer resumes
- 

## 8. Common Synchronization Problems

- **Starvation** → thread never gets CPU
  - **Deadlock** → threads wait forever
  - **Race condition** → inconsistent data
- 

## Most Common MCQ Traps


- wait(), notify() are in Object class
  - wait() must be inside synchronized
  - sleep() does not release lock
  - Deadlock has 4 conditions
  - notify() wakes only one thread
  - notifyAll() wakes all threads
- 

## One-Line Last-Minute Revision

- Synchronization → lock
  - Deadlock → infinite wait
  - wait() → release lock
  - notify() → wake one
  - notifyAll() → wake all
  - Producer-Consumer → wait & notify
- 

If you want next:

- ✓ **60-80 MCQs** from Sessions 28 & 29
- ✓ **Deadlock & wait/notify tricky MCQs**
- ✓ **Complete Multithreading MCQ set**
- ✓ **Sessions 1-29 full revision**

Just tell me 

## Session 30: Generics and Reflection API

Lecture: □ Introduction to generics □ Generic classes □ Generic methods □ Wild cards (upper and lower) □ Metadata & Reflection

Below are **short, exam-oriented, MCQ-focused notes** for **Session 30 – Generics & Reflection API**, written in **easy language, point-wise**, and covering **all common MCQ traps**.

# Session 30 – Generics & Reflection API (MCQ Focused Notes)

## 1. Introduction to Generics

**Generics** allow you to write **type-safe and reusable code** by using **parameterized types**.

```
java

List<Integer> list = new ArrayList<>();
```

### Advantages

- **Compile-time type checking**
- No explicit type casting
- Code reusability

### MCQ Points

- Generics work at **compile time**
- Introduced in **Java 5**
- Improve **type safety**

## 2. Generic Classes

A class that works with **any data type**.

```
java

class Box<T> {
    T value;
}
```

### MCQ Points

- T is a **type parameter**
- Common letters: T, E, K, V

- Cannot use primitive types directly (`int` ❌, `Integer` ✓)

### 3. Generic Methods

A method that introduces its **own type parameter**.

```
java

public static <T> void display(T data) {
    System.out.println(data);
}
```

#### MCQ Points

- Type parameter declared **before return type**
- Can be static or non-static
- Independent of class generics

### 4. Bounded Types

Restrict the type parameter to a **specific class or interface**.

```
java

class Test<T extends Number> { }
```

#### MCQ Points

- **extends** keyword used
- Works for **class & interface**
- Multiple bounds allowed (class first)

### 5. Wildcards in Generics

Wildcards are represented using ?.

#### A. Upper Bounded Wildcard

```
java

List<? extends Number>
```

- ✓ Accepts Number and its subclasses
- ❌ Cannot add elements (except null)

## 🔑 MCQ Point

- Used for **read-only**

### B. Lower Bounded Wildcard

```
java
List<? super Integer>
```

- ✓ Accepts Integer and its superclasses
- ✓ Can add Integer objects

## 🔑 MCQ Point

- Used for **write-only**

### C. Unbounded Wildcard

```
java
List<?>
```

- ✓ Accepts any type
- ✗ Cannot add elements

## 6. extends vs super (MCQ Favorite)

Keyword	Purpose	Use
<b>extends</b>	Upper bound	Read
<b>super</b>	Lower bound	Write

## 👉 PECS Rule

Producer → extends

Consumer → super

## 7. Type Erasure

- Generic information is **removed at runtime**
- JVM sees only **raw types**

## 🔑 MCQ Points

- Generics are **compile-time only**

- No runtime generic info

## 8. Metadata in Java

**Metadata** = information about data (class, method, fields).

- Stored using **annotations**
- Used by tools & frameworks

### MCQ Points

- Metadata is not part of business logic
- Used by Reflection API

## 9. Reflection API

Reflection allows **examining and modifying classes at runtime**.

### Main Classes

- **Class**
- **Method**
- **Field**
- **Constructor**

```
java
```

```
Class c = Class.forName("Student");
```

### MCQ Points

- Part of **java.lang.reflect**
- Used in frameworks (Spring, Hibernate)
- Slower than normal code

## 10. Uses of Reflection

- Access private members
- Load classes dynamically
- Framework development

### MCQ Points

- Can break encapsulation
- Powerful but risky

## 11. Generics vs Non-Generics (MCQ)

Feature	Generics	Non-Generics
Type Safety	✓	✗
Casting	✗	✓
Errors	Compile-time	Runtime

## Most Common MCQ Traps


- Generics do not support primitives
- Wildcards use ?
- extends → read, super → write
- Type erasure removes generic info
- Reflection works at runtime
- Reflection can access private members

## One-Line Last-Minute Revision

- Generics → type safety
- Generic class → reusable
- Generic method → independent
- Wildcards → flexibility
- extends → upper bound
- super → lower bound
- Reflection → runtime access

If you want next:

- ✓ **40-60 MCQs** from Session 30
- ✓ **Generics tricky wildcard MCQs**
- ✓ **Reflection API MCQs**
- ✓ **Sessions 1-30 full MCQ revision**

Just tell me 

give me the mcq from each section or each prompt that i have given you give atleast 20 mcq questions for each session

Understood 👍

Below are **MCQs (exam-oriented, no explanations)** for **EACH SESSION** you asked, **minimum 20 MCQs per session**.

These are **direct MCQ-type questions** useful for **competitive / semester exams**.

## ✅ SESSION 11 – Access Modifiers, Packages, Constructors

### MCQs (20)

1. Which access modifier allows access everywhere?  
A) private B) default C) protected D) public
2. Which modifier is accessible within same package only?  
A) public B) protected C) default D) private
3. Which modifier allows access in subclass outside package?  
A) private B) protected C) default D) final
4. Which access modifier has highest restriction?  
A) protected B) default C) private D) public
5. Default access means:  
A) public B) protected C) no keyword D) private
6. Package statement must be written at:  
A) End of file  
B) Anywhere  
C) First line  
D) After import
7. Which package is auto-imported?  
A) java.util B) java.io C) java.lang D) java.sql
8. Which keyword is used for static import?  
A) import B) static C) import static D) package
9. Static import is used for:  
A) objects B) constructors C) static members D) classes
10. Constructor chaining within same class uses:  
A) super() B) this() C) new D) finalize()
11. Parent constructor is called using:  
A) this() B) parent() C) super() D) base()
12. this() must be:  
A) last statement  
B) anywhere

- C) first statement
- D) optional
- 13. Can this() and super() be used together?  
A) Yes B) No C) Sometimes D) Depends
- 14. Protected members outside package are accessed using:  
A) object reference  
B) subclass reference  
C) static block  
D) package name
- 15. Which constructor cannot be accessed outside class?  
A) public B) protected C) default D) private
- 16. Package improves:  
A) speed B) security C) memory D) compilation
- 17. Which is invalid access?  
A) private in same class  
B) protected in subclass  
C) default outside package  
D) public anywhere
- 18. Static import reduces:  
A) memory B) execution time C) code length D) errors
- 19. Can class be private?  
A) Yes B) No C) Sometimes D) Depends
- 20. Protected members are accessed via:  
A) object B) inheritance C) interface D) package only

## **SESSION 12 – Garbage Collection**

### **MCQs (20)**

- 1. Garbage collection is handled by:  
A) Programmer B) JVM C) OS D) Compiler
- 2. GC works on:  
A) stack B) method area C) heap D) registers
- 3. Which method requests GC?  
A) finalize() B) delete() C) gc() D) free()
- 4. System.gc() means:  
A) Force GC  
B) Request GC  
C) Stop GC  
D) Disable GC
- 5. Object becomes eligible for GC when:  
A) program ends



- B) reference removed
  - C) memory full
  - D) finalize called
6. Which makes object eligible for GC?
- A) Null reference
  - B) Reassign reference
  - C) Island of isolation
  - D) All
7. Island of isolation means:
- A) circular reference
  - B) object without reference
  - C) static object
  - D) null object
8. finalize() is called by:
- A) compiler B) JVM C) programmer D) OS
9. finalize() is guaranteed to execute?
- A) Yes B) No C) Sometimes D) Always
10. finalize() is called:
- A) before GC
  - B) after GC
  - C) during GC
  - D) at end
11. GC destroys:
- A) variables B) references C) objects D) classes
12. Java has destructor?
- A) Yes B) No C) finalize() D) delete
13. finalize() introduced in:
- A) Java 1.0 B) Java 5 C) Java 8 D) Java 11
14. GC thread priority is:
- A) max B) min C) normal D) same
15. Circular references prevent GC?
- A) Yes B) No C) Sometimes D) Depends
16. Runtime.gc() is:
- A) force GC
  - B) request GC
  - C) stop GC
  - D) free memory
17. GC frees memory from:
- A) stack B) heap C) registers D) cache
18. finalize() is:
- A) constructor
  - B) destructor

- C) normal method
- D) GC callback
- 19. GC occurs when:
  - A) object created
  - B) JVM decides
  - C) programmer calls
  - D) always
- 20. Which is deprecated?
  - A) finalize() B) gc() C) Object D) Runtime

## **SESSION 13-14 – Wrapper & String**

### MCQs (20)

1. Wrapper classes are in:
  - A) java.util B) java.io C) java.lang D) java.sql
2. Wrapper classes are:
  - A) mutable B) immutable C) static D) abstract
3. Integer constant pool range:
  - A) 0-255 B) -128 to 127 C) -256 to 255 D) all
4. Autoboxing introduced in:
  - A) Java 1.0 B) Java 1.2 C) Java 5 D) Java 8
5. String is:
  - A) mutable B) immutable C) synchronized D) abstract
6. StringBuilder is:
  - A) thread-safe B) slow C) immutable D) not thread-safe
7. StringBuffer is:
  - A) fast B) unsafe C) thread-safe D) immutable
8. String pool stores:
  - A) objects B) literals C) arrays D) classes
9. == compares:
  - A) content B) reference C) value D) memory
10. equals() compares:
  - A) reference B) content C) hash D) address
11. new String("Java") creates:
  - A) 1 object
  - B) 2 objects
  - C) 0 objects
  - D) 3 objects
12. StringBuilder introduced in:
  - A) Java 1.0 B) Java 5 C) Java 7 D) Java 8

13. Wrapper for int is:  
A) Int B) Integer C) Long D) Short
  14. Character pool range:  
A) 0-255 B) 0-127 C) 1-128 D) all
  15. Boolean pool contains:  
A) 1 value B) 2 values C) 3 values D) many
  16. intern() returns:  
A) heap ref B) pool ref C) new object D) null
  17. String class is:  
A) abstract B) final C) interface D) static
  18. StringBuilder is faster because:  
A) mutable B) no sync C) small D) pool
  19. Which is mutable?  
A) String B) Integer C) StringBuffer D) Character
  20. Wrapper classes used in:  
A) arrays B) collections C) loops D) GC
- 

## **SESSION 15-16 – Exception Handling**

### **MCQs (20)**

1. Parent of all exceptions:  
A) Object B) Throwable C) Error D) Exception
2. Checked exception is checked at:  
A) runtime B) compile time C) execution D) load
3. RuntimeException is:  
A) checked B) unchecked C) error D) fatal
4. IOException is:  
A) checked B) unchecked C) error D) runtime
5. finally block executes:  
A) always B) sometimes C) never D) optional
6. finally doesn't execute when:  
A) exception  
B) return  
C) System.exit()  
D) try block
7. throw keyword is used to:  
A) declare exception  
B) create exception  
C) handle exception  
D) ignore exception

8. throws is used in:
  - A) method body
  - B) class body
  - C) method signature
  - D) try block
9. Multi-catch introduced in:
  - A) Java 5 B) Java 6 C) Java 7 D) Java 8
10. Which cannot be in multi-catch?
  - A) siblings
  - B) unrelated
  - C) parent-child
  - D) exceptions
11. User-defined checked exception extends:
  - A) Throwable B) Error C) Exception D) RuntimeException
12. Unchecked exception extends:
  - A) Exception B) Error C) RuntimeException D) Throwable
13. Exception propagation applies to:
  - A) checked
  - B) unchecked
  - C) errors
  - D) all
14. try block must have:
  - A) finally
  - B) catch or finally
  - C) catch
  - D) throw
15. catch block parameter type must be:
  - A) primitive
  - B) object
  - C) Throwable
  - D) String
16. Order of catch blocks should be:
  - A) parent first
  - B) child first
  - C) any order
  - D) random
17. Which is Error?
  - A) IOException
  - B) SQLException
  - C) StackOverflowError
  - D) NullPointerException
18. throw throws how many exceptions?
  - A) many B) zero C) one D) two

19. Which is unchecked?  
A) ClassNotFoundException  
B) SQLException  
C) ArithmeticException  
D) IOException
20. Exception handling improves:  
A) speed  
B) readability  
C) reliability  
D) memory

⚠️ **Due to message size limits**, I will continue with **Sessions 17–30 MCQs in the NEXT MESSAGE**.  
👉 Reply **“Continue”** and I’ll send **remaining sessions MCQs without delay**.

continue

Perfect 👍

Continuing with **20 MCQs for EACH remaining session** exactly as per your syllabus.

## ✅ **SESSION 17 – java.io & java.nio, Serialization**

### **MCQs (20)**

1. InputStream is used for:  
A) character input B) byte input C) object input D) text input
2. OutputStream deals with:  
A) characters B) bytes C) strings D) objects
3. Reader class is for:  
A) byte data B) binary data C) character data D) object data
4. Writer class handles:  
A) byte stream B) character stream C) network stream D) object stream
5. FileInputStream reads data in:  
A) characters B) objects C) bytes D) strings
6. Buffered streams improve:  
A) security B) speed C) memory D) GC
7. NIO stands for:  
A) New Input Output  
B) Network IO  
C) Non-blocking IO  
D) Native IO

8. NIO package introduced in:  
A) Java 1.2 B) Java 1.4 C) Java 5 D) Java 7
9. Core NIO classes are in:  
A) java.io B) java.util C) java.nio D) java.net
10. Serialization means:  
A) file writing  
B) object to byte stream  
C) byte to object  
D) cloning
11. Interface used for serialization:  
A) Cloneable  
B) Serializable  
C) Externalizable  
D) Runnable
12. transient keyword prevents:  
A) compilation  
B) GC  
C) serialization  
D) inheritance
13. static variables are:  
A) serialized  
B) not serialized  
C) copied  
D) cloned
14. Deserialization converts:  
A) file to stream  
B) stream to object  
C) object to class  
D) memory to disk
15. serialVersionUID is used for:  
A) security  
B) version control  
C) speed  
D) GC
16. Externalizable provides:  
A) default serialization  
B) custom serialization  
C) no serialization  
D) fast IO
17. Shallow copy copies:  
A) primitive only  
B) reference only

- C) deep objects
- D) memory
- 18. Deep copy copies:
  - A) reference
  - B) original object
  - C) entire object graph
  - D) address
- 19. Which supports non-blocking IO?
  - A) IO
  - B) NIO
  - C) Reader
  - D) Writer
- 20. Which class writes objects?
  - A) FileWriter
  - B) BufferedWriter
  - C) ObjectOutputStream
  - D) PrintWriter

## **SESSION 18 – Object Class & Date/Time**

### MCQs (20)

1. Parent class of all classes:
  - A) Class B) System C) Object D) Runtime
2. toString() belongs to:
  - A) String B) Object C) System D) Class
3. equals() is used to compare:
  - A) reference
  - B) memory
  - C) content
  - D) hash
4. hashCode() returns:
  - A) boolean B) object C) integer D) long
5. Contract of equals() and hashCode():
  - A) optional
  - B) must match
  - C) ignored
  - D) runtime
6. Overriding toString() improves:
  - A) speed B) readability C) memory D) GC
7. Date class is in:
  - A) java.time B) java.util C) java.sql D) java.text

8. Date represents:
  - A) only date
  - B) only time
  - C) both
  - D) timezone
9. Calendar class is:
  - A) final B) abstract C) static D) interface
10. SimpleDateFormat is in:
  - A) java.util B) java.sql C) java.text D) java.time
11. Pattern "dd-MM-yyyy" means:
  - A) year-month-day
  - B) day-month-year
  - C) month-day-year
  - D) time format
12. Parsing converts:
  - A) Date to String
  - B) String to Date
  - C) Date to Calendar
  - D) Calendar to String
13. format() method converts:
  - A) String → Date
  - B) Date → String
  - C) Calendar → Date
  - D) String → Calendar
14. equals() overridden in:
  - A) Object
  - B) Date
  - C) String
  - D) All
15. hashCode() used in:
  - A) IO
  - B) threading
  - C) collections
  - D) GC
16. Calendar.getInstance() returns:
  - A) object
  - B) Date
  - C) Calendar
  - D) String
17. Object class is in:
  - A) java.util
  - B) java.lang



- C) java.io
- D) java.time
- 18. Default toString() prints:
  - A) value
  - B) content
  - C) classname@hash
  - D) memory
- 19. If equals() is overridden, must override:
  - A) toString()
  - B) finalize()
  - C) hashCode()
  - D) clone()
- 20. java.time introduced in:
  - A) Java 5 B) Java 6 C) Java 7 D) Java 8

## **SESSIONS 19–22 – Collection Framework (List, Queue)**

### MCQs (20)

1. Root interface of collections:
  - A) List B) Set C) Map D) Collection
2. List allows:
  - A) duplicates B) no order C) no index D) sorted only
3. ArrayList is:
  - A) synchronized
  - B) dynamic array
  - C) linked list
  - D) map
4. LinkedList stores elements as:
  - A) array B) tree C) nodes D) hash
5. Vector is:
  - A) non-sync B) sync C) fast D) new
6. ArrayList default capacity:
  - A) 5 B) 8 C) 10 D) 16
7. LinkedList implements:
  - A) List
  - B) Queue
  - C) Deque
  - D) All
8. Vector is slower because:
  - A) GC
  - B) synchronization

- C) size
  - D) memory
9. Collections.sort() works on:
- A) Map
  - B) List
  - C) Set
  - D) Queue
10. Comparable is in:
- A) java.lang
  - B) java.util
  - C) java.io
  - D) java.sql
11. Comparator provides:
- A) natural order
  - B) custom order
  - C) default order
  - D) fixed order
12. compareTo() returns:
- A) boolean
  - B) int
  - C) object
  - D) long
13. Queue follows:
- A) LIFO
  - B) FIFO
  - C) random
  - D) sorted
14. poll() removes:
- A) last
  - B) random
  - C) head
  - D) tail
15. peek() returns:
- A) removes element
  - B) exception
  - C) null
  - D) head
16. PriorityQueue order is:
- A) insertion
  - B) natural
  - C) random
  - D) reverse
17. Iterator is used to:
- A) modify

- B) traverse
  - C) sort
  - D) delete
18. Enumeration is used with:
- A) ArrayList
  - B) LinkedList
  - C) Vector
  - D) HashSet
19. ListIterator allows:
- A) forward only
  - B) backward only
  - C) both
  - D) none
20. Fail-fast iterator throws:
- A) IOException
  - B) SQLException
  - C) ConcurrentModificationException
  - D) RuntimeException

## **SESSIONS 23–26 – Set, Map, Streams**

### MCQs (20)

1. Set allows duplicates?  
A) Yes B) No C) Sometimes D) Depends
2. HashSet uses:  
A) array B) tree C) hash table D) list
3. LinkedHashSet maintains:  
A) sorting  
B) insertion order  
C) random  
D) natural
4. TreeSet stores elements in:  
A) insertion order  
B) random  
C) sorted order  
D) reverse
5. TreeSet uses:  
A) hash  
B) comparator  
C) equals  
D) index
6. Backed collection means:  
A) independent

- B) view of collection
  - C) static
  - D) immutable
7. Map stores data as:
- A) values
  - B) keys
  - C) key-value
  - D) index-value
8. HashMap allows:
- A) null key
  - B) multiple null keys
  - C) no null
  - D) sorted
9. Hashtable allows null?
- A) key only
  - B) value only
  - C) both
  - D) none
10. TreeMap sorts by:
- A) value
  - B) key
  - C) insertion
  - D) index
11. LinkedHashMap maintains:
- A) random
  - B) sorted
  - C) insertion order
  - D) reverse
12. ConcurrentHashMap is:
- A) non-sync
  - B) thread-safe
  - C) slow
  - D) old
13. Streams introduced in:
- A) Java 5 B) Java 6 C) Java 7 D) Java 8
14. Stream is used for:
- A) storage
  - B) processing
  - C) IO
  - D) thread
15. filter() is:
- A) terminal
  - B) intermediate

- C) static
  - D) final
16. map() returns:
- A) list
  - B) stream
  - C) set
  - D) array
17. forEach() is:
- A) intermediate
  - B) terminal
  - C) lazy
  - D) static
18. Parallel stream improves:
- A) memory
  - B) speed
  - C) size
  - D) GC
19. Collectors.toList() returns:
- A) set
  - B) map
  - C) list
  - D) array
20. Optional is used to avoid:
- A) IO error
  - B) null pointer
  - C) GC
  - D) exception

---

## **SESSION 27 – Multithreading Basics**

### **MCQs (20)**

1. Thread is created using:
- A) Runnable only
  - B) Thread only
  - C) both
  - D) none
2. start() calls:
- A) run() directly
  - B) JVM
  - C) OS
  - D) GC
3. run() executes:
- A) new thread

- B) main thread
  - C) JVM thread
  - D) GC thread
4. sleep() pauses:
- A) object
  - B) thread
  - C) JVM
  - D) CPU
5. join() makes thread:
- A) stop
  - B) wait
  - C) sleep
  - D) dead
6. yield() gives chance to:
- A) same thread
  - B) GC
  - C) other threads
  - D) OS
7. Thread priority range:
- A) 0-10
  - B) 1-10
  - C) 1-5
  - D) 0-5
8. Default priority:
- A) 1 B) 3 C) 5 D) 10
9. MAX\_PRIORITY value:
- A) 1 B) 5 C) 8 D) 10
10. MIN\_PRIORITY value:
- A) 1 B) 0 C) 5 D) 10
11. ThreadGroup groups:
- A) objects
  - B) classes
  - C) threads
  - D) packages
12. main thread priority is:
- A) 1 B) 3 C) 5 D) 10
13. Daemon thread is:
- A) user thread
  - B) background thread
  - C) main thread
  - D) dead thread
14. setDaemon() must be called:
- A) anytime

- B) after start
  - C) before start
  - D) in run
15. JVM exits when:
- A) main ends
  - B) all user threads end
  - C) daemon ends
  - D) GC ends
16. Thread class is in:
- A) java.lang
  - B) java.util
  - C) java.io
  - D) java.concurrent
17. Runnable is:
- A) class
  - B) abstract class
  - C) interface
  - D) method
18. Thread safety relates to:
- A) IO
  - B) memory
  - C) concurrency
  - D) GC
19. Which method stops thread safely?
- A) stop()
  - B) destroy()
  - C) interrupt()
  - D) exit()
20. stop() method is:
- A) safe
  - B) recommended
  - C) deprecated
  - D) new



## SESSIONS 28-29 – Synchronization & Deadlock

### MCQs (20)

1. Synchronization prevents:
- A) deadlock
  - B) race condition
  - C) GC
  - D) memory leak

2. synchronized keyword locks:
  - A) code
  - B) method
  - C) object
  - D) class
3. Object-level lock is on:
  - A) class
  - B) method
  - C) object
  - D) thread
4. Class-level lock is on:
  - A) object
  - B) thread
  - C) class
  - D) CPU
5. Deadlock occurs when:
  - A) two threads wait forever
  - B) thread sleeps
  - C) thread stops
  - D) GC runs
6. Deadlock condition includes:
  - A) mutual exclusion
  - B) hold and wait
  - C) circular wait
  - D) all
7. wait() releases:
  - A) CPU
  - B) lock
  - C) thread
  - D) memory
8. sleep() releases lock?
  - A) Yes B) No C) Sometimes D) Depends
9. notify() wakes:
  - A) all threads
  - B) one thread
  - C) main thread
  - D) GC
10. notifyAll() wakes:
  - A) one
  - B) none
  - C) all
  - D) main



11. wait(), notify() belong to:
  - A) Thread
  - B) Runnable
  - C) Object
  - D) System
12. Producer-consumer uses:
  - A) sleep
  - B) join
  - C) wait-notify
  - D) yield
13. synchronized block is faster than method?
  - A) Yes B) No C) Same D) Depends
14. Deadlock is detected by:
  - A) compiler
  - B) JVM
  - C) OS
  - D) programmer
15. wait() must be called inside:
  - A) method
  - B) loop
  - C) synchronized block
  - D) try
16. notify() must be called inside:
  - A) main
  - B) synchronized block
  - C) finally
  - D) constructor
17. Which releases lock?
  - A) sleep()
  - B) wait()
  - C) join()
  - D) yield()
18. Deadlock solution includes:
  - A) resource ordering
  - B) timeout
  - C) avoid nested locks
  - D) all
19. notifyAll avoids:
  - A) GC
  - B) starvation
  - C) exception
  - D) IO

20. Thread communication uses:

- A) sleep
- B) interrupt
- C) wait-notify
- D) join

## **SESSION 30 – Generics & Reflection**

### MCQs (20)

1. Generics introduced in:

- A) Java 1.2 B) Java 1.4 C) Java 5 D) Java 8

2. Generics provide:

- A) speed
- B) memory
- C) type safety
- D) GC

3. Generic class syntax:

- A) class A<T>
- B) class A
- C) class <T>A
- D) A<T>

4. Wildcard symbol is:

- A) \* B) ? C) # D) &

5. Upper bounded wildcard:

- A) <? super T> B) <? extends T> C) <?>
- D) <T>

6. Lower bounded wildcard:

- A) <? extends T> B) <? super T> C) <?>
- D) <T>

7. Generic method declared using:

- A) <T> before return
- B) after method
- C) inside body
- D) none

8. Type erasure occurs at:

- A) runtime
- B) compile time
- C) load time
- D) execution

9. Reflection API is in:

- A) java.util
- B) java.lang.reflect

- C) java.io
  - D) java.sql
10. Reflection allows:
- A) runtime inspection
  - B) compile check
  - C) GC
  - D) IO
11. Class object obtained by:
- A) new
  - B) Class.forName()
  - C) getClass()
  - D) all
12. Method class is in:
- A) java.util
  - B) java.lang
  - C) java.lang.reflect
  - D) java.io
13. Reflection breaks:
- A) encapsulation
  - B) inheritance
  - C) polymorphism
  - D) abstraction
14. Generics cannot use:
- A) wrapper types
  - B) primitives
  - C) classes
  - D) interfaces
15. Raw type means:
- A) no generics
  - B) dynamic
  - C) object
  - D) runtime
16. instanceof works with generics?
- A) Yes B) No C) Sometimes D) Depends
17. Reflection is slower because of:
- A) GC
  - B) runtime checks
  - C) memory
  - D) IO
18. Annotation metadata read using:
- A) IO
  - B) Reflection
  - C) Stream
  - D) Thread

19. getDeclaredMethods() returns:





- A) public methods
- B) private only
- C) all methods
- D) static methods


20. Generics improve:


- A) readability
- B) maintainability
- C) safety
- D) all

## All sessions COMPLETED

If you want:

-  **Answer keys**
-  **Important MCQs only**
-  **Interview-based MCQs**
-  **PDF / Word format**

Just tell me 

 Answer keys

Perfect 

Below are the **ANSWER KEYS** for all the MCQs I gave you, **session-wise**, clean and exam-ready.

## **SESSION 17 – java.io, java.nio, Serialization**

1. B
2. B
3. C
4. B
5. C
6. B
7. C
8. B
9. C
10. B
11. B

- 12. C
- 13. B
- 14. B
- 15. B
- 16. B
- 17. B
- 18. C
- 19. B
- 20. C

---

## **SESSION 18 – Object Class & Date/Time**

- 1. C
- 2. B
- 3. C
- 4. C
- 5. B
- 6. B
- 7. B
- 8. C
- 9. B
- 10. C
- 11. B
- 12. B
- 13. B
- 14. D
- 15. C
- 16. C
- 17. B
- 18. C
- 19. C
- 20. D

---

## **SESSIONS 19–22 – Collection Framework (List & Queue)**

- 1. D

2. A
  3. B
  4. C
  5. B
  6. C
  7. D
  8. B
  9. B
  10. A
  11. B
  12. B
  13. B
  14. C
  15. D
  16. B
  17. B
  18. C
  19. C
  20. C
- 

## SESSIONS 23–26 – Set, Map & Streams

1. B
2. C
3. B
4. C
5. B
6. B
7. C
8. A
9. D
10. B
11. C
12. B
13. D

- 14. B
  - 15. B
  - 16. B
  - 17. B
  - 18. B
  - 19. C
  - 20. B
- 

## **SESSION 27 – Multithreading Basics**

- 1. C
  - 2. B
  - 3. B
  - 4. B
  - 5. B
  - 6. C
  - 7. B
  - 8. C
  - 9. D
  - 10. A
  - 11. C
  - 12. C
  - 13. B
  - 14. C
  - 15. B
  - 16. A
  - 17. C
  - 18. C
  - 19. C
  - 20. C
- 

## **SESSIONS 28–29 – Synchronization & Deadlock**

- 1. B
- 2. C
- 3. C

4. C
  5. A
  6. D
  7. B
  8. B
  9. B
  10. C
  11. C
  12. C
  13. A
  14. D
  15. C
  16. B
  17. B
  18. D
  19. B
  20. C
- 





## **SESSION 30 – Generics & Reflection**

1. C
2. C
3. A
4. B
5. B
6. B
7. A
8. B
9. B
10. A
11. D
12. C
13. A
14. B
15. A



- 16. B
  - 17. B
  - 18. B
  - 19. C
  - 20. D
- 

### **NEXT OPTIONS (Highly Recommended for Exam)**

-  **Most repeated MCQs only**
-  **Tricky MCQs with explanations**
-  **PDF for last-day revision**
-  **Mock test (100 MCQs, timed)**

Just tell me what you want 