Core java

Index

📘 JAVA – Chapter 1 : Introduction to Java

🔹 **Java** म्हणजे काय**?**

🇮🇳 मराठी**:**

Java ही **High-Level, Object-Oriented Programming Language** आहे.
Java चा वापर software, website, mobile app (Android) आणि enterprise application बनवण्यासाठी होतो.
ही language 1995 मध्ये **Sun Microsystems** (आता Oracle) यांनी तयार केली.

🇬🇧 **English:**

Java is a **high-level, object-oriented programming language**.
It is used to develop software, websites, Android mobile applications, and enterprise systems.
Java was developed in 1995 by **Sun Microsystems** (now Oracle).

🔹 **Java** चा इतिहास **(History of Java)**

🇮🇳 मराठी**:**

Java ची निर्मिती **James Gosling** आणि त्यांच्या team ने केली.
सुरुवातीला या language चे नाव **Oak** होते.
हा project **Green Project** म्हणून ओळखला जात होता.
नंतर Oak चे नाव बदलून **Java** ठेवण्यात आले.

🇬🇧 **English:**

Java was developed by **James Gosling** and his team.
Initially, the language was named **Oak**.
It was part of the **Green Project**.
Later, the name was changed to **Java**.

☕ Java ची वैशिष्ट्ये (Features of Java)

1️⃣ Simple

🔹 मराठी:

Java ची syntax सोपी आहे.
C++ प्रमाणे pointers आणि operator overloading Java मध्ये नाही, त्यामुळे शिकायला सोपी आहे.

🔹 **English:**

Java has a simple and easy-to-understand syntax.
It does not support pointers and operator overloading like C++, which makes it easier to learn.

2️⃣ Object-Oriented

🔹 मराठी:

Java पूर्णपणे **Object-Oriented Programming (OOP)** वर आधारित आहे.
Java मध्ये class, object, inheritance, polymorphism, encapsulation आणि abstraction वापरले जातात.

🔹 **English:**

Java is completely based on **Object-Oriented Programming (OOP)** concepts.
It uses class, object, inheritance, polymorphism, encapsulation, and abstraction.

3️⃣ Platform Independent

🔹 मराठी:

Java ही **Platform Independent** language आहे.
Java code एकदाच लिहिला की तो कोणत्याही operating system वर चालतो.
यालाच **Write Once, Run Anywhere (WORA)** म्हणतात.

🔹 **English:**

Java is a **platform-independent** language.
Once written, Java code can run on any operating system.
This feature is known as **Write Once, Run Anywhere (WORA)**.

4️⃣ Secure

🔹 **मराठी:**

Java सुरक्षित आहे कारण:

- Pointer वापर नाही
- Bytecode verification असते
- Direct memory access नाही

🔹 **English:**

Java is secure because:

- It does not use pointers
- It has bytecode verification
- It does not allow direct memory access

5️⃣ Robust

🔹 **मराठी:**

Java मजबूत (Robust) आहे कारण:

- Strong memory management आहे
- Exception handling support आहे
- Garbage collection आहे

🔹 **English:**

Java is robust because:

- It has strong memory management

- It supports exception handling
- It provides garbage collection

☕ Java Platform Components

☕ मराठी:

Java Platform मध्ये 3 मुख्य भाग असतात:

1. **JDK (Java Development Kit)** – Program develop करण्यासाठी
2. **JRE (Java Runtime Environment)** – Program run करण्यासाठी
3. **JVM (Java Virtual Machine)** – Bytecode execute करण्यासाठी

☕ English:

The Java platform consists of three main components:

1. **JDK (Java Development Kit)** – Used to develop Java programs
2. **JRE (Java Runtime Environment)** – Used to run Java programs
3. **JVM (Java Virtual Machine)** – Executes Java bytecode

☕ Java Program Execution Process

☕ मराठी:

1. `.java` file लिहिली जाते

2. Compiler source code ला `.class` bytecode मध्ये convert करतो
3. JVM bytecode execute करते
4. Output मिळतो

🇬🇧 English:

1. A `.java` file is written
2. The compiler converts it into `.class` bytecode
3. The JVM executes the bytecode
4. Output is produced

✅Chapter 1 Summary

🇮🇳 मराठी:

- Java OOP language आहे
- Platform independent आहे

Secure आणि robust आहे

🇬🇧 English:

- Java is an object-oriented language
- It is platform independent
- It is secure and robust

Chapter 2 : Java Programming Basics

☕ JAVA – Chapter 2 : Java Programming Basics

☕ Java Program Structure

☕ मराठी:

Java program मध्ये खालील गोष्टी असतात:

- class declaration
- main method
- statements

```
class Demo {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

☕ English:

A Java program consists of:

- class declaration
- main method
- statements

```
class Demo {
    public static void main(String[] args) {
        System.out.println("Hello Java");
```

```
    }
}
```

🔹 Variables

🇮🇳 मराठी:

Variable म्हणजे **data store** करण्यासाठी वापरलेली **memory location**.

```
int a = 10;
```

🌍 English:

A variable is a **memory location used to store data**.

```
int a = 10;
```

🔹 Data Types in Java

🇮🇳 मराठी:

Java मध्ये 2 प्रकारचे data types असतात:

1. Primitive
2. Non-Primitive

🌍 English:

Java has two types of data types:

1. Primitive

2. Non-Primitive

🔁 Control Statements

🇮🇳 मराठी:

Program चा flow control करण्यासाठी वापरले जातात:

- if, if-else
- switch
- for, while, do-while

🇬🇧 English:

Control statements are used to control the flow of a program:

- if, if-else
- switch
- for, while, do-while

✅Chapter 2 Summary

🇮🇳 मराठी:

- Java syntax basics
- Variables आणि data types
- Control statements

? English:

- Basic Java syntax
- Variables and data types
- Control statements

Chapter 3 : Object Oriented

☐ JAVA – Chapter 3 : Object Oriented Programming (OOP) Concepts

---

☐ OOP म्हणजे काय?

☐ मराठी:

Object Oriented Programming (OOP) ही programming पद्धत आहे िजथे program **objects** आण **classes** वर आधा रत असतो.
Java ही पूणपणे OOP concepts follow करणारी language आहे.

OOP मुळे:

- Code reusable होतो
- Program secure होतो
- Maintenance सोपे होते

☐ English:

Object Oriented Programming (OOP) is a programming approach where a program is based on **objects and classes**.
Java follows OOP concepts completely.

Benefits of OOP:

- Code reusability
- Better security
- Easy maintenance

❓ Class म्हणजे काय?

❓ मराठी:

Class म्हणजे **object** तयार करण्यासाठीचा **blueprint (**आराखडा**)** आहे.
Class मध्ये variables आणि methods define केलेले असतात.

```
class Student {
    int id;
    String name;
}
```

❓ English:

A **class is a blueprint used to create objects**.
It contains variables and methods.

```
class Student {
    int id;
    String name;
}
```

❓ Object म्हणजे काय?

❓ मराठी:

Object हा **class** चा **instance** असतो**.**
Object memory मध्ये real existence ठेवतो.

```
Student s1 = new Student();
```

❓ English:

An **object is an instance of a class**.
It represents a real-world entity and occupies memory.

```
Student s1 = new Student();
```

🔒 Encapsulation

🇮🇳 मराठी:

Encapsulation म्हणजे **data** आणि **methods** एकत्र बांधून ठेवणे.
Data hiding साठी `private` keyword वापरला जातो.

🇬🇧 English:

Encapsulation means **wrapping data and methods together into a single unit**.
Data hiding is achieved using the `private` keyword.

🧬 Inheritance

🇮🇳 मराठी:

Inheritance म्हणजे एक **class** दुसऱ्या **class** चे **properties inherit** करतो.
`extends` keyword वापरला जातो.

```
class A { }
class B extends A { }
```

🇬🇧 English:

Inheritance allows **one class to acquire properties of another class**.
The `extends` keyword is used.

```
class A { }
class B extends A { }
```

□ Polymorphism

□ मराठी:

Polymorphism म्हणजे एकाच नावाचे वेगवेगळे रूप.
Java मध्ये दोन प्रकार:

1. Method Overloading
2. Method Overriding

□ English:

Polymorphism means **many forms of a single method**.
Two types in Java:

1. Method Overloading
2. Method Overriding

□ Abstraction

□ मराठी:

Abstraction म्हणजे आवश्यक माहिती दाखवणे आणि उरलेली लपवणे. Java
मध्ये abstraction:

- Abstract class
- Interface

□ English:

Abstraction means **showing essential details and hiding implementation**.
Achieved using:

- Abstract classes
- Interfaces

✅Chapter 3 Summary

🇮🇳 मराठी:

- OOP concepts Java चा core आहेत
- Class आणि Object main components आहेत
- Encapsulation, Inheritance, Polymorphism, Abstraction महत्वाचे

🇬🇧 English:

- OOP concepts are the core of Java
- Class and object are main components
- Encapsulation, inheritance, polymorphism, abstraction are important

Operators in Java

☐ Operators in Java

☐ Operator म्हणजे काय?

☐ मराठी:

Operator म्हणजे असे चिन्ह (symbol) जे एका किंवा अधिक **operands** वर **operation** करतात.
Java मध्ये operators चा वापर calculation, comparison, logical decision आणि bit-level operation साठी होतो.

☐ English:

An **operator** is a symbol that **performs an operation on one or more operands**.
In Java, operators are used for calculation, comparison, logical decisions, and bit-level operations.

☐ Java मधील Operators चे प्रकार

1️⃣ Arithmetic Operators

☐ मराठी:

Arithmetic operators गणितीय **calculation** साठी वापरले जातात.

Operator	काम
+	Addition
-	Subtraction
*	Multiplication
/	Division

%	Modulus

```
int a = 10, b = 3;
System.out.println(a + b); // 13
System.out.println(a % b); // 1
```

🇬🇧 English:

Arithmetic operators are used for **mathematical calculations**.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

```
int a = 10, b = 3;
System.out.println(a + b); // 13
System.out.println(a % b); // 1
```

2️⃣ Relational (Comparison) Operators

🇮🇳 मराठी:

हे operators दोन **values compare** करतात आणि result `true` किंवा `false` देतात.

Operator	अर्थ
>	Greater than
<	Less than
>=	Greater than equal

<=	Less than equal
==	Equal
!=	Not equal

```
System.out.println(a > b);  // true
```

🇬🇧 English:

Relational operators **compare two values** and return `true` or `false`.

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

```
System.out.println(a > b);  // true
```

3️⃣ Logical Operators

🇮🇳 मराठी:

Logical operators **conditions combine** करण्यासाठी वापरले जातात.

Operator	अर्थ
&&	Logical AND
!	Logical NOT

```
System.out.println(a > 5 && b < 5); // true
```

🇬🇧 English:

Logical operators are used to **combine multiple conditions**.

Operator	Meaning
&&	Logical AND
!	Logical NOT

```
System.out.println(a > 5 && b < 5); // true
```

4️⃣ Assignment Operators

🇮🇳 मराठी:

Assignment operators **value assign** करण्यासाठी वापरले जातात.

Operator	Example
=	a = 10
+=	a += 5
-=	a -= 2
*=	a *= 3
/=	a /= 2

🇬🇧 English:

Assignment operators are used to **assign values to variables**.

Operator	Example

=	a = 10
+=	a += 5
-=	a -= 2
*=	a *= 3
/=	a /= 2

5️⃣ Unary Operators

🔹 मराठी:

Unary operators एकाच **operand** वर काम करतात.

Operator	अर्थ
++	Increment
--	Decrement
!	Logical NOT

```
int x = 5;
x++;
```

🔹 English:

Unary operators work on **a single operand**.

Operator	**Meaning**
++	Increment
--	Decrement
!	Logical NOT

```
int x = 5;
x++;
```

6️⃣ Bitwise Operators

🇮🇳 मराठी:

Bitwise operators **bit level operation** साठी वापरले जातात.

Operator	अर्थ
&	Bitwise AND
^	XOR
~	Bitwise NOT
<<	Left Shift
>>	Right Shift

🇬🇧 English:

Bitwise operators perform **bit-level operations**.

Operator	**Meaning**
&	Bitwise AND
^	XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift

7️⃣ Ternary Operator

🇮🇳 मराठी:

Ternary operator हा **short form of if-else** आहे.

```
int max = (a > b) ? a : b;
```

🇬🇧 English:

The ternary operator is a **short form of if-else**.

```
int max = (a > b) ? a : b;
```

8️⃣ instanceof Operator

🇮🇳 मराठी:

`instanceof` operator **object** कोणत्या **class** चा आहे हे तपासतो.

```
System.out.println(obj instanceof Student);
```

🇬🇧 English:

The `instanceof` operator checks **whether an object belongs to a specific class**.

```
System.out.println(obj instanceof Student);
```

✅ Operators Summary

🇮🇳 मराठी:

- Operators program मध्ये logic build करतात
- Java मध्ये विविध प्रकारचे operators आहेत
- Exam आणि interview साठी खूप महत्वाचे topic

🇬🇧 English:

- Operators help build logic in programs
- Java provides different types of operators
- Very important topic for exams and interviews

Decision Making Statements

🔥 Decision Making Statements in Java

🔥 Decision Making म्हणजे काय?

🔥 मराठी:

Decision Making Statements वापरून program **condition true** की **false** यावर **decision** घेतो.
Condition च्या result वर program चा flow बदलतो.

Java मधील decision making statements:

- if
- if-else
- if-else-if ladder
- switch

🔥 English:

Decision Making Statements allow a program to **make decisions based on conditions**.
The flow of the program changes depending on whether the condition is true or false.

Decision making statements in Java:

- if
- if-else
- if-else-if ladder
- switch

1️⃣ if Statement

🔹 मराठी:

`if` statement एक **condition check** करते.
Condition true असेल तरच code execute होतो.

Syntax:

```
if (condition) {
    // code
}
```

Example:

```
int age = 20;
if (age >= 18) {
    System.out.println("Eligible for vote");
}
```

🔹 English:

The `if` statement **checks a condition**.
If the condition is true, the code inside the block executes.

Syntax:

```
if (condition) {
    // code
}
```

Example:

```
int age = 20;
if (age >= 18) {
    System.out.println("Eligible for vote");
}
```

2️⃣ if-else Statement

🇮🇳 मराठी:

`if-else` statement मध्ये दोन **blocks** असतात.
Condition true असेल तर `if` block, false असेल तर `else` block execute होतो.

Syntax:

```
if (condition) {
    // if block
} else {
    // else block
}
```

Example:

```
int num = 5;
if (num % 2 == 0) {
    System.out.println("Even number");
} else {
    System.out.println("Odd number");
}
```

🇬🇧 English:

The `if-else` statement has **two blocks**.
If the condition is true, the `if` block executes; otherwise, the `else` block executes.

Syntax:

```
if (condition) {
    // if block
} else {
    // else block
}
```

Example:

```
int num = 5;
if (num % 2 == 0) {
    System.out.println("Even number");
} else {
    System.out.println("Odd number");
}
```

3️⃣ if-else-if Ladder

🔹 मराठी:

`if-else-if ladder` **multiple conditions check** करण्यासाठी वापरली जाते.
पहिली true condition आढळताच बाकीच्या conditions skip होतात.

Syntax:

```
if (condition1) {
    // code
} else if (condition2) {
    // code
} else {
    // code
}
```

Example:

```
int marks = 75;
if (marks >= 80) {
    System.out.println("Distinction");
} else if (marks >= 60) {
    System.out.println("First Class");
} else {
    System.out.println("Pass");
}
```

🔹 English:

The `if-else-if ladder` is used to **check multiple conditions**.
Once a condition is true, the remaining conditions are skipped.

Syntax:

```
if (condition1) {
    // code
} else if (condition2) {
    // code
} else {
    // code
}
```

Example:

```
int marks = 75;
if (marks >= 80) {
    System.out.println("Distinction");
} else if (marks >= 60) {
    System.out.println("First Class");
} else {
    System.out.println("Pass");
}
```

---

4 Nested if

मराठी:

Nested if म्हणजे `if` च्या आत अजून एक `if` statement असते.
Complex conditions साठी वापरले जाते.

Example:

```
int age = 20;
int weight = 55;

if (age >= 18) {
    if (weight >= 50) {
```

```
        System.out.println("Eligible");
    }
}
```

🇬🇧 English:

A **nested if** means an `if` statement inside another `if`.
It is used for complex decision making.

Example:

```
int age = 20;
int weight = 55;

if (age >= 18) {
    if (weight >= 50) {
        System.out.println("Eligible");
    }
}
```

5️⃣ switch Statement

🇮🇳 मराठी:

`switch` statement **multiple choices** मधून एक **execute** करते.
`break` वापरले नाही तर fall-through होतो.

Syntax:

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
```

```
        // code
}
```

Example:

```
int day = 2;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Invalid day");
}
```

🇬🇧 English:

The `switch` statement executes **one block from multiple choices**.
The `break` statement prevents fall-through.

Syntax:

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
}
```

Example:

```
int day = 2;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Invalid day");
}
```

❓ if-else vs switch

❓ मराठी:

- if-else complex conditions साठी योग्य
- switch fixed values साठी योग्य

❓ English:

- if-else is suitable for complex conditions
- switch is suitable for fixed values

✅Decision Making Summary

❓ मराठी:

- Decision making program flow control करते

- if, if-else, ladder, switch वापरले जातात
- Exam आणि interview साठी महत्वाचा topic

🇬🇧 English:

- Decision making controls program flow
- if, if-else, ladder, switch are used
- Important topic for exams and interviews

Looping Statements

🔁 Looping Statements in Java

🔁 Loop म्हणजे काय?

🔁 मराठी:

Loop म्हणजे एखादा code block पुन्हा पुन्हा **execute** करण्यासाठी वापरलेली **structure**.
जेव्हा एखादं काम multiple times करायचं असतं, तेव्हा loop वापरतात.

Java मधील loops:

- for loop
- while loop
- do-while loop

🔁 English:

A **loop** is a structure used to **execute a block of code repeatedly**.
Loops are used when a task needs to be performed multiple times.

Loops in Java:

- for loop
- while loop
- do-while loop

1️⃣ for Loop

🔁 मराठी:

`for loop` तेव्हा वापरतात जेव्हा **iterations (repeat count)** आधीच माहीत असते.

Syntax:

```
for(initialization; condition; increment/decrement) {
    // code
}
```

Example:

```
for(int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

🇬🇧 English:

The `for loop` is used when the **number of iterations is known in advance**.

Syntax:

```
for(initialization; condition; increment/decrement) {
    // code
}
```

Example:

```
for(int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

2️⃣ while Loop

🇮🇳 मराठी:

`while loop` तेव्हा वापरतात जेव्हा **iterations** किती वेळा होणार आहेत हे माहीत नसते.
पहिले condition check होते, मग loop execute होतो.

Syntax:

```
while(condition) {
    // code
}
```

Example:

```
int i = 1;
while(i <= 5) {
    System.out.println(i);
    i++;
}
```

🇬🇧 English:

The `while loop` is used when the **number of iterations is not known**.
The condition is checked first, then the loop executes.

Syntax:

```
while(condition) {
    // code
}
```

Example:

```
int i = 1;
while(i <= 5) {
    System.out.println(i);
    i++;
}
```

3️⃣ do-while Loop

🇮🇳 मराठी:

`do-while loop` मध्ये **code** किमान एकदा **execute** होतो,
कारण condition नंतर check केली जाते.

Syntax:

```
do {
    // code
} while(condition);
```

Example:

```
int i = 1;
do {
    System.out.println(i);
    i++;
} while(i <= 5);
```

🇬🇧 English:

In a `do-while loop`, the **code executes at least once**,
because the condition is checked after execution.

Syntax:

```
do {
    // code
} while(condition);
```

Example:

```
int i = 1;
do {
    System.out.println(i);
    i++;
} while(i <= 5);
```

? while vs do-while

? मराठी:

- while → condition आधी check
- do-while → code आधी execute

? English:

- while → condition is checked first
- do-while → code executes first

? Nested Loop

? मराठी:

Nested loop म्हणजे एका **loop** च्या आत दुसरा **loop**.
Patterns, tables, matrix साठी वापरले जातात.

Example:

```
for(int i = 1; i <= 3; i++) {
    for(int j = 1; j <= 3; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
```

? English:

A **nested loop** means **one loop inside another loop**.
Used for patterns, tables, and matrices.

Example:

```
for(int i = 1; i <= 3; i++) {
    for(int j = 1; j <= 3; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
```

🔁 Loop Control Statements

🛑 break Statement

🇮🇳 मराठी:

`break` loop ताबडतोब थांबवतो.

```
for(int i = 1; i <= 5; i++) {
    if(i == 3) break;
    System.out.println(i);
}
```

🇬🇧 English:

The `break` statement **terminates the loop immediately**.

```
for(int i = 1; i <= 5; i++) {
    if(i == 3) break;
    System.out.println(i);
}
```

⏭️ continue Statement

🇮🇳 मराठी:

`continue` current iteration skip करतो आणि पुढच्या iteration ला जातो.

```
for(int i = 1; i <= 5; i++) {
    if(i == 3) continue;
    System.out.println(i);
}
```

🇬🇧 English:

The `continue` statement skips the current iteration and moves to the next one.

```
for(int i = 1; i <= 5; i++) {
    if(i == 3) continue;
    System.out.println(i);
}
```

🆚 for vs while

🇮🇳 मराठी:

- for → iterations माहीत असतील तेव्हा
- while → iterations माहीत नसतील तेव्हा

🇬🇧 English:

- for → when iterations are known
- while → when iterations are unknown

✅Looping Summary

🇮🇳 मराठी:

- Loop repeated execution साठी वापरतात
- Java मध्ये for, while, do-while loops आहेत
- break आणि continue loop control करतात

🇬🇧 English:

- Loops are used for repeated execution
- Java supports for, while, and do-while loops
- break and continue control loop execution

OOP

Polymorphism in Java

🔹 Polymorphism in Java

🔹 Polymorphism म्हणजे काय?

🔹 मराठी:

Polymorphism म्हणजे एकाच नावाची वेगवेगळी रूपे.
Java मध्ये polymorphism मुळे एक **method** वेगवेगळ्या प्रकारे काम करू शकतो.

उदाहरण:
`+` operator numbers साठी addition करतो, strings साठी concatenation करतो.

🔹 English:

Polymorphism means **many forms of a single entity**.
In Java, polymorphism allows **a method to perform different tasks**.

Example:
The `+` operator performs addition for numbers and concatenation for strings.

🔹 Polymorphism चे प्रकार

🔹 मराठी:

Java मध्ये polymorphism चे दोन प्रकार आहेत:

1. Compile Time Polymorphism
2. Runtime Polymorphism

🔹 English:

There are **two types of polymorphism** in Java:

1. Compile-time polymorphism
2. Runtime polymorphism

1️⃣ Compile Time Polymorphism (Method Overloading)

🔹 मराठी:

Compile time polymorphism ला **Method Overloading** म्हणतात.
एका class मध्ये:

- Method नाव same असते
- पण parameters वेगळे असतात (number, type किंवा order)

🔹 Return type change करून overloading होत नाही.

Example:

```
class Demo {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

🔹 English:

Compile-time polymorphism is achieved using **method overloading**.
In the same class:

- Method name is the same
- Parameters must be different (number, type, or order)

 Changing only the return type does not achieve overloading.

Example:

```
class Demo {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

---

2 Runtime Polymorphism (Method Overriding)

मराठी:

Runtime polymorphism ला **Method Overriding** म्हणतात.
Subclass मध्ये parent class ची method **same signature** ने **redefine** केली जाते.
Method call runtime ला decide होतो.

 Inheritance आवश्यक आहे.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
```

```
    }
}
```

🔹 English:

Runtime polymorphism is achieved using **method overriding**.
A subclass provides its own implementation of a parent class method with the **same method signature**.
The method call is resolved at runtime.

🔹 Inheritance is required.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
```

🔹 Overloading vs Overriding

🔹 मराठी:

Method Overloading	Method Overriding
Same class	Parent–child class
Compile time	Runtime
Parameters different	Same parameters
Inheritance not required	Inheritance required

🇬🇧 English:

Method Overloading	Method Overriding
Same class	Parent–child class
Compile time	Runtime
Different parameters	Same parameters
No inheritance required	Inheritance required

🔹 Polymorphism चे फायदे

🇮🇳 मराठी:

- Code reusability वाढते
- Flexibility मिळते
- Loose coupling तयार होते

🇬🇧 English:

- Improves code reusability
- Provides flexibility
- Enables loose coupling

🔹 Method Overriding चे Rules

🇮🇳 मराठी:

- Method नाव आणि parameters same असावेत
- Access modifier कमी restrict करू शकत नाही
- `final`, `static`, `private` methods override होत नाहीत

🇬🇧 English:

- Method name and parameters must be the same
- Access modifier cannot be more restrictive
- `final`, `static`, and `private` methods cannot be overridden

🔁 Dynamic Method Dispatch

🇮🇳 मराठी:

Parent class reference आणि child class object वापरून
runtime polymorphism achieve होते.

```
Animal a = new Dog();
a.sound();
```

🇬🇧 English:

Runtime polymorphism is achieved using **dynamic method dispatch**,
where a parent class reference points to a child class object.

```
Animal a = new Dog();
a.sound();
```

✅Polymorphism Summary

🇮🇳 मराठी:

- Polymorphism म्हणजे one name, many forms
- Overloading → compile time
- Overriding → runtime
- Java OOP चा खूप महत्वाचा concept

🇬🇧 English:

- Polymorphism means one name, many forms
- Overloading → compile time
- Overriding → runtime
- Very important OOP concept in Java

□ Class and Object in Java

🔹 Class and Object in Java

🔹 Class म्हणजे काय?

🔹 मराठी:

Class म्हणजे **object** तयार करण्यासाठीचा **blueprint (**आराखडा**)** आहे.
Class मध्ये:

- Variables (data members)
- Methods (functions)

define केलेले असतात.
Class स्वतः memory occupy करत नाही.

Example:

```
class Student {
    int id;
    String name;

    void display() {
        System.out.println(id + " " + name);
    }
}
```

🔹 English:

A **class is a blueprint used to create objects**.
A class contains:

- Variables (data members)
- Methods (functions)

A class does not occupy memory by itself.

Example:

```
class Student {
    int id;
    String name;

    void display() {
        System.out.println(id + " " + name);
    }
}
```

? Object म्हणजे काय?

? मराठी:

Object हा **class** चा **instance** असतो.
Object memory मध्ये actual space occupy करतो.
Object च्या माध्यमातून class मधील members access केले जातात.

Example:

```
Student s1 = new Student();
s1.id = 1;
s1.name = "Amit";
s1.display();
```

? English:

An **object is an instance of a class**.
An object occupies memory.
Class members are accessed using objects.

Example:

```
Student s1 = new Student();
s1.id = 1;
```

```
s1.name = "Amit";
s1.display();
```

🔹 Class आणि Object मधील फरक

🔹 मराठी:

Class	Object
Blueprint आहे	Real entity आहे
Memory घेत नाही	Memory घेतो
Logical entity	Physical entity
One time defined	Multiple objects बनतात

🔹 English:

Class	Object
Blueprint	Real-world entity
Does not occupy memory	Occupies memory
Logical entity	Physical entity
Defined once	Multiple objects can be created

🔹 Object कसा तयार करतात?

🔹 मराठी:

Object `new` keyword वापरून तयार करतात.

Syntax:

```
ClassName obj = new ClassName();
```

🇬🇧 English:

An object is created using the `new` keyword.

Syntax:

```
ClassName obj = new ClassName();
```

🇮🇳 Constructor म्हणजे काय?

🇮🇳 मराठी:

Constructor हा **special method** आहे:

- Class नावासारखाच नाव असतो
- Object create होताच automatically call होतो
- Values initialize करण्यासाठी वापरला जातो

Example:

```
class Student {
    int id;
    Student(int i) {
        id = i;
    }
}
```

🇬🇧 English:

A constructor is a **special method**:

- Same name as the class

- Automatically called when an object is created
- Used to initialize values

Example:

```
class Student {
    int id;
    Student(int i) {
        id = i;
    }
}
```

? this keyword

? मराठी:

`this` keyword **current object** ला **refer** करतो.
Local variable आणि instance variable differentiate करण्यासाठी वापरतात.

? English:

The `this` keyword **refers to the current object**.
It is used to differentiate between local and instance variables.

? Object चे प्रकार

? मराठी:

Java मध्ये object चे प्रकार:

- Anonymous object
- Named object

🇬🇧 English:

Types of objects in Java:

- Anonymous object
- Named object

🎯 Object Oriented Programming मध्ये Class–Object चे महत्व

🇮🇳 मराठी:

- Code organized राहतो
- Reusability वाढते
- Real-world problem easily solve होतात

🇬🇧 English:

- Helps organize code
- Increases reusability
- Solves real-world problems efficiently

✅Class and Object Summary

🇮🇳 मराठी:

- Class blueprint आहे
- Object real entity आहे
- Object memory घेतो
- Constructor object initialize करतो

? English:

- Class is a blueprint
- Object is a real entity
- Object occupies memory
- Constructor initializes the object

□ Array of Objects in Java

☕ Array of Objects in Java

❓ Array of Object म्हणजे काय?

🇮🇳 मराठी:

Array of Objects म्हणजे एका **class** च्या अनेक **objects** ला **array** मध्ये **store** करणे.
जेव्हा आपल्याला एकाच प्रकारच्या अनेक objects सोबत काम करायचं असतं, तेव्हा array of objects वापरतात.

उदा: Student class चे अनेक students.

🇬🇧 English:

An **array of objects** means **storing multiple objects of the same class in an array**.
It is used when we need to work with many objects of the same type.

Example: multiple student objects.

❓ Array of Objects का वापरतात?

🇮🇳 मराठी:

- Multiple objects easily manage करता येतात
- Code organized राहतो
- Data processing सोपे होते

🇬🇧 English:

- Helps manage multiple objects easily
- Keeps code organized
- Simplifies data processing

🔹 Array of Objects कसा declare करतात?

🔹 मराठी:

पहिले **class** तयार करतात, मग त्याचा array declare करतात.

Syntax:

```
ClassName[] arrayName = new ClassName[size];
```

🔹 English:

First, a **class is created**, then an array of that class is declared.

Syntax:

```
ClassName[] arrayName = new ClassName[size];
```

🔹 Array of Objects – Example

🔹 मराठी:

खाली Student class चा array of objects दाखवला आहे.

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
```

```
        this.name = name;
    }

    void display() {
        System.out.println(id + " " + name);
    }
}

class Test {
    public static void main(String[] args) {
        Student[] s = new Student[3];

        s[0] = new Student(1, "Amit");
        s[1] = new Student(2, "Rahul");
        s[2] = new Student(3, "Neha");

        for(int i = 0; i < s.length; i++) {
            s[i].display();
        }
    }
}
```

? English:

Below is an example of an **array of Student objects**.

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    void display() {
        System.out.println(id + " " + name);
    }
```

```
}

class Test {
    public static void main(String[] args) {
        Student[] s = new Student[3];

        s[0] = new Student(1, "Amit");
        s[1] = new Student(2, "Rahul");
        s[2] = new Student(3, "Neha");

        for(int i = 0; i < s.length; i++) {
            s[i].display();
        }
    }
}
```

🔹 for-each loop वापरून Array of Objects

🔹 मराठी:

Array of objects access करण्यासाठी **enhanced for loop** वापरू शकतो.

```
for(Student st : s) {
    st.display();
}
```

🔹 English:

An **enhanced for loop** can also be used to access an array of objects.

```
for(Student st : s) {
    st.display();
}
```

❓ Array of Objects vs Normal Array

❓ मराठी:

Normal Array	Array of Objects
Primitive values store	Object references store
int, float, char	Class objects
Simple data	Complex data

❓ English:

Normal Array	Array of Objects
Stores primitive values	Stores object references
int, float, char	Class objects
Simple data	Complex data

❓ Real-Life Example

❓ मराठी:

College system मध्ये:

- अनेक Students
- अनेक Teachers

हे सगळे **array of objects** वापरून store करता येतात.

❓ English:

In a college system:

- Multiple students
- Multiple teachers

All can be stored using **arrays of objects**.

🎯 Interview Points (खूप महत्वाचे)

🔹 मराठी:

- Array of objects reference store करतो
- प्रत्येक index ला object assign करावा लागतो
- `new` keyword वापरून object तयार होतो

🔹 English:

- An array of objects stores references
- Each index must be assigned an object
- Objects are created using the `new` keyword

✅ Array of Objects Summary

🔹 मराठी:

- Array of objects = multiple objects in one array
- Complex data handle करण्यासाठी उपयुक्त
- OOP concepts मध्ये खूप वापर

English:

- Array of objects stores multiple objects
- Useful for handling complex data
- Widely used in OOP concepts

Class in Java – Interview Questions

☐ Class in Java – Interview Questions

Q1 Class म्हणजे काय?

☐ मराठी:

Class म्हणजे **object** तयार करण्यासाठीचा **blueprint (**आराखडा**)** आहे.
Class मध्ये variables आणि methods असतात.
Class स्वतः memory occupy करत नाही.

☐ English:

A **class is a blueprint used to create objects**.
It contains variables and methods.
A class itself does not occupy memory.

Q2 Class आणि Object मधील फरक काय?

☐ मराठी:

Class हा logical entity आहे, तर object हा real entity आहे.
Class एकदाच define करतात, object अनेक तयार होऊ शकतात.

☐ English:

A class is a logical entity, whereas an object is a real entity.
A class is defined once, but many objects can be created from it.

Q3 Java मध्ये class कसा declare करतात?

🇮🇳 मराठी:

`class` keyword वापरून class declare करतात.

```
class Student {
    int id;
}
```

🇬🇧 English:

A class is declared using the `class` keyword.

```
class Student {
    int id;
}
```

Q4 Class memory कधी घेतो?

🇮🇳 मराठी:

Class **memory** घेत नाही.
Class चा object तयार केल्यावर memory allocate होते.

🇬🇧 English:

A class does **not occupy memory**.
Memory is allocated only when an object of the class is created.

Q5 Java मध्ये एका file मध्ये किती classes असू शकतात?

🇮🇳 मराठी:

एका Java file मध्ये **multiple classes** असू शकतात,
पण फक्त एकच **public class** असू शकतो.

🇬🇧 English:

A Java file can contain **multiple classes**,
but **only one public class** is allowed.

Q6 **Public class** चं नाव **file** नावासारखं का असतं**?**

🇮🇳 मराठी**:**

Java compiler ला public class **directly access** करायची असते,
म्हणून public class आणि file नाव same असतं.

🇬🇧 English:

The public class must match the file name because
the Java compiler accesses the public class directly.

Q7 **Class** मध्ये काय काय असू शकतं**?**

🇮🇳 मराठी**:**

Class मध्ये:

- Variables
- Methods
- Constructors
- Blocks
- Inner classes

असू शकतात.

🇬🇧 English:

A class can contain:

- Variables
- Methods
- Constructors
- Blocks
- Inner classes

Q8 Static class म्हणजे काय?

🇮🇳 मराठी:

Java मध्ये **top-level static class** नसते.
फक्त **inner class static** असू शकते.

🇬🇧 English:

Java does **not allow static top-level classes**.
Only **inner classes can be static**.

Q9 Final class म्हणजे काय?

🇮🇳 मराठी:

Final class inherit करता येत नाही.
उदा: `String` class.

🇬🇧 English:

A **final class cannot be inherited**.
Example: `String` class.

Q🇮🇳 Abstract class म्हणजे काय?

🇮🇳 मराठी:

Abstract class मध्ये:

- Abstract methods असू शकतात
- Object तयार करता येत नाही
- Inheritance साठी वापरतात

🇬🇧 English:

An abstract class:

- Can contain abstract methods
- Cannot be instantiated
- Is used for inheritance

Q11 Class आणि Structure (C) मधील फरक?

🇮🇳 मराठी:

Class मध्ये data + methods दोन्ही असतात,
Structure मध्ये फक्त data असतो.

🇬🇧 English:

A class contains both data and methods,
whereas a structure contains only data.

Q12 Class OOP मध्ये का महत्वाची आहे?

🇮🇳 मराठी:

- Code organize राहतो
- Reusability वाढते
- Real-world modeling सोपे होते

🇬🇧 English:

- Helps organize code
- Improves reusability
- Simplifies real-world modeling

🎯 Top Interview One-Liners (खूप उपयोगी)

🇮🇳 मराठी:

- Class is a blueprint
- Object is an instance of class
- Class does not occupy memory
- Public class name = file name

🇬🇧 English:

- Class is a blueprint
- Object is an instance of a class
- Class does not occupy memory
- Public class name must match file name

✅Class Interview Summary

🇮🇳 मराठी:

- Class Java OOP चा core आहे
- Interview मध्ये basic + conceptual प्रश्न विचारले जातात

🇬🇧 English:

- Class is the core of Java OOP
- Interviews focus on basic and conceptual questions

□ Constructor in Java

☐ Constructor in Java

☐ Constructor म्हणजे काय?

☐ मराठी:

Constructor हा एक **special method** आहे जो **object create** होताच **automatically call** होतो.
Constructor चा उपयोग **object** ला **initial values** देण्यासाठी केला जातो.

Constructor ची वैशिष्ट्ये:

- Constructor चं नाव class सारखंच असतं
- Constructor ला return type नसतो (void सुद्धा नाही)
- `new` keyword वापरल्यावर constructor call होतो

☐ English:

A **constructor is a special method** that is **automatically called when an object is created**.
It is used to **initialize an object**.

Features of a constructor:

- Constructor name is the same as the class name
- It has no return type (not even void)
- It is invoked when the `new` keyword is used

☐ Constructor का वापरतात?

☐ मराठी:

- Object initialization साठी
- Default values assign करण्यासाठी
- Object creation वेळेस setup करण्यासाठी

🇬🇧 English:

- To initialize objects
- To assign default values
- To perform setup at the time of object creation

🇮🇳 Constructor चे प्रकार

🇮🇳 मराठी:

Java मध्ये constructor चे **3** प्रकार आहेत:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor (User-defined)

🇬🇧 English:

Java has **three types of constructors**:

1. Default constructor
2. Parameterized constructor

3. Copy constructor (user-defined)

1️⃣ Default Constructor

🔹 मराठी:

Default constructor:

- कोणतेही parameters नसतात
- Compiler automatically देतो (जर आपण constructor लिहिला नसेल तर)
- Instance variables ला default values देतो

Example:

```
class Demo {
    int a;

    Demo() {
        System.out.println("Default Constructor");
    }
}
```

🔹 English:

A default constructor:

- Has no parameters
- Is provided by the compiler if no constructor is defined
- Initializes instance variables with default values

Example:

```
class Demo {
```

```
    int a;

    Demo() {
        System.out.println("Default Constructor");
    }
}
```

2️⃣ Parameterized Constructor

🇮🇳 मराठी:

Parameterized constructor:

- Parameters घेतो
- Values pass करून object initialize करतो

Example:

```
class Student {
    int id;
    String name;

    Student(int i, String n) {
        id = i;
        name = n;
    }
}
```

🇬🇧 English:

A parameterized constructor:

- Accepts parameters
- Initializes objects using passed values

Example:

```
class Student {
    int id;
    String name;

    Student(int i, String n) {
        id = i;
        name = n;
    }
}
```

3️⃣ Copy Constructor

🇮🇳 मराठी:

Copy constructor:

- एका object ची values दुसऱ्या object मध्ये copy करतो
- Java मध्ये default नसतो (आपण स्वतः define करावा लागतो)

Example:

```
class Student {
    int id;

    Student(Student s) {
        id = s.id;
    }
}
```

🇬🇧 English:

A copy constructor:

- Copies values from one object to another
- Not provided by Java by default (user-defined)

Example:

```
class Student {
    int id;

    Student(Student s) {
        id = s.id;
    }
}
```

□ this Keyword Constructor मध्ये

□ मराठी:

`this` keyword:

- Current object ला refer करतो
- Local आणि instance variables distinguish करतो

```
Student(int id) {
    this.id = id;
}
```

□ English:

The `this` keyword:

- Refers to the current object
- Differentiates local and instance variables

```
Student(int id) {
    this.id = id;
}
```

🔹 Constructor Overloading

🇮🇳 मराठी:

एकाच class मध्ये **multiple constructors** असू शकतात,
फक्त parameters वेगळे असायला हवेत.

🇬🇧 English:

A class can have **multiple constructors** with different parameters.
This is called constructor overloading.

🔹 Constructor vs Method

🇮🇳 मराठी:

Constructor	Method
Class नावासारखं नाव	कोणतंही नाव
Return type नाही	Return type असतो
Object creation ला call	Explicit call

🇬🇧 English:

Constructor	Method
Same name as class	Any name
No return type	Has return type

Called during object creation Called explicitly

🔹 Constructor Important Rules (Interview)

🔹 मराठी:

- Constructor inherit होत नाही
- Constructor override होत नाही
- Constructor private असू शकतो
- Abstract constructor नसतो

🔹 English:

- Constructors are not inherited
- Constructors cannot be overridden
- A constructor can be private
- Constructors cannot be abstract

✅Constructor Summary

🔹 मराठी:

- Constructor object initialize करतो
- Types: default, parameterized, copy
- OOP मध्ये खूप महत्वाचा concept

🇬🇧 English:

- Constructors initialize objects
- Types: default, parameterized, copy
- Very important OOP concept

□ Inheritance in Java

🔹 Inheritance in Java

🔹 Inheritance म्हणजे काय?

🔹 मराठी:

Inheritance म्हणजे एका **class** ने दुसऱ्या **class** चे **properties** आणि **methods** घेणे. जो class properties देतो त्याला **Parent / Super class** म्हणतात आणि जो class घेतो त्याला **Child / Sub class** म्हणतात.

Inheritance मुळे:

- Code reusability वाढते
- Program maintain करणे सोपे होते
- Method overriding शक्य होते

🔹 English:

Inheritance is a mechanism where **one class acquires the properties and methods of another class**.
The class that provides properties is called the **Parent / Super class**, and the class that receives them is called the **Child / Sub class**.

Inheritance helps in:

- Code reusability
- Easy maintenance
- Method overriding

🔹 Inheritance साठी वापरले जाणारे Keywords

🇮🇳 मराठी:

Java मध्ये inheritance साठी:

- `extends` → class inherit करण्यासाठी
- `implements` → interface inherit करण्यासाठी

🇬🇧 English:

In Java, inheritance uses:

- `extends` → to inherit a class
- `implements` → to inherit an interface

🔹 Simple Inheritance (Single Inheritance)

🇮🇳 मराठी:

Single inheritance मध्ये एक child class फक्त एका **parent class** कडून **inherit** करते.

```
class Animal {
    void eat() {
        System.out.println("Eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking");
    }
}
```

🇬🇧 English:

In **single inheritance**, a child class inherits from **only one parent class**.

```
class Animal {
    void eat() {
        System.out.println("Eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking");
    }
}
```

🇮🇳 Inheritance चे प्रकार

🇮🇳 मराठी:

Java मध्ये inheritance चे प्रकार:

1. Single
2. Multilevel
3. Hierarchical

❌ **Multiple inheritance (classes** मध्ये**) Java support** करत नाही.

🇬🇧 English:

Types of inheritance in Java:

1. Single
2. Multilevel

3. Hierarchical

 Java does not support multiple inheritance using classes.

---

1 Multilevel Inheritance

मराठी:

एका class कडून दुसरा class, आणि त्याच्याकडून तिसरा class inherit करतो.

```
class A {
    void showA() {}
}

class B extends A {
    void showB() {}
}

class C extends B {
    void showC() {}
}
```

English:

In multilevel inheritance, a class inherits from another class,
and a third class inherits from the second class.

```
class A {
    void showA() {}
}

class B extends A {
    void showB() {}
}

class C extends B {
```

```
    void showC() {}
}
```

2️⃣ Hierarchical Inheritance

🇮🇳 मराठी:

एका parent class पासून **multiple child classes** inherit करतात.

```
class Animal { }

class Dog extends Animal { }

class Cat extends Animal { }
```

🇬🇧 English:

In hierarchical inheritance, **multiple child classes inherit from a single parent class**.

```
class Animal { }

class Dog extends Animal { }

class Cat extends Animal { }
```

❓ Multiple Inheritance Java मध्ये का नाही?

🇮🇳 मराठी:

Java classes मध्ये multiple inheritance support करत नाही कारण:

- **Diamond problem** येतो
- Ambiguity (confusion) तयार होते

 पण **interfaces** वापरून **multiple inheritance** शक्य आहे.

English:

Java does not support multiple inheritance using classes because:

- It causes the **diamond problem**
- It creates ambiguity

 However, multiple inheritance is possible using **interfaces**.

---

super Keyword

मराठी:

`super` keyword वापरून:

- Parent class चा constructor call करतात
- Parent class चे variable/method access करतात

```
super();
super.show();
```

English:

The `super` keyword is used to:

- Call the parent class constructor
- Access parent class variables/methods

```
super();
super.show();
```

🔹 Method Overriding (Inheritance मध्ये)

🔹 मराठी:

Child class मध्ये parent class ची method **same signature** ने **redefine** केली जाते,
यालाच **method overriding** म्हणतात.

🔹 English:

Method overriding occurs when a child class **redefines a parent class method with the same signature**.

🔹 Inheritance चे फायदे

🔹 मराठी:

- Code duplication कमी होते
- Reusability वाढते
- Polymorphism support मिळतो

🔹 English:

- Reduces code duplication
- Improves reusability
- Supports polymorphism

🔹 Inheritance चे नियम (Interview Important)

🔹 मराठी:

- Constructor inherit होत नाही
- Private members inherit होत नाहीत (directly)
- Final class inherit होत नाही

🔹 English:

- Constructors are not inherited
- Private members are not directly inherited
- A final class cannot be inherited

✅Inheritance Summary

🔹 मराठी:

- Inheritance parent-child relationship तयार करते
- extends keyword वापरतात
- Java मध्ये 3 inheritance types आहेत

🔹 English:

- Inheritance creates a parent-child relationship
- Uses the extends keyword
- Java supports three types of inheritance

□ Function (Method) in Java

❓ Function (Method) in Java

❓ Function म्हणजे काय?

❓ मराठी:

Java मध्ये **Function** ला **Method** म्हणतात.
Method म्हणजे एखादं ठराविक काम करणारा **code** चा **block**.
Method मुळे:

- Code पुन्हा पुन्हा वापरता येतो (reusability)
- Program easy आणि readable होतो

❓ English:

In Java, a **function is called a method**.
A method is a **block of code that performs a specific task**.
Methods help in:

- Code reusability
- Improving readability and structure

❓ Method का वापरतात?

❓ मराठी:

- Code duplication कमी करण्यासाठी
- Large program छोटे भागात divide करण्यासाठी

- Debugging सोपं करण्यासाठी

🇬🇧 English:

- To reduce code duplication
- To divide large programs into smaller parts
- To make debugging easier

🔹 Method ची Syntax

🇮🇳 मराठी:

```
returnType methodName(parameters) {
    // method body
}
```

🇬🇧 English:

```
returnType methodName(parameters) {
    // method body
}
```

🔹 Method चे भाग (Parts of Method)

🇮🇳 मराठी:

1. Return type – method काय value return करणार
2. Method name – method चं नाव

3. Parameters – input values
4. Method body – actual code

🇬🇧 English:

1. Return type – value returned by the method
2. Method name – name of the method
3. Parameters – input values
4. Method body – actual code

🇮🇳 Method चे प्रकार

🇮🇳 मराठी:

Java मध्ये method चे **4 common types** आहेत:

1. No parameter, no return type
2. Parameter with no return type
3. No parameter with return type
4. Parameter with return type

🇬🇧 English:

Java has **four common types of methods**:

1. No parameter, no return type

2. Parameter with no return type
3. No parameter with return type
4. Parameter with return type

1️⃣ No Parameter, No Return Type

🇮🇳 मराठी:

Method ना input घेत नाही आणि output देत नाही.

```
void show() {
    System.out.println("Hello");
}
```

🇬🇧 English:

The method does not take input and does not return output.

```
void show() {
    System.out.println("Hello");
}
```

2️⃣ Parameter with No Return Type

🇮🇳 मराठी:

Method input घेतो पण value return करत नाही.

```
void add(int a, int b) {
    System.out.println(a + b);
}
```

🔹 English:

The method takes input but does not return any value.

```
void add(int a, int b) {
    System.out.println(a + b);
}
```

3️⃣ No Parameter with Return Type

🔹 मराठी:

Method input घेत नाही पण value return करतो.

```
int getNumber() {
    return 10;
}
```

🔹 English:

The method does not take input but returns a value.

```
int getNumber() {
    return 10;
}
```

4️⃣ Parameter with Return Type

🔹 मराठी:

Method input घेतो आणि value return करतो.

```
int multiply(int a, int b) {
    return a * b;
}
```

🇬🇧 English:

The method takes input and returns a value.

```
int multiply(int a, int b) {
    return a * b;
}
```

🔧 Method Call (Function Call)

🇮🇳 मराठी:

Method call करण्यासाठी **object** वापरतात
(जर method static नसेल तर).

```
Demo d = new Demo();
d.show();
```

🇬🇧 English:

To call a method, an **object is required**
(if the method is not static).

```
Demo d = new Demo();
d.show();
```

🔒 Static Method

🇮🇳 मराठी:

`static` method:

- Class नावाने call करता येतो

- Object ची गरज नसते

```
static void display() {
    System.out.println("Static Method");
}
```

🇬🇧 English:

A `static` method:

- Can be called using the class name
- Does not require an object

```
static void display() {
    System.out.println("Static Method");
}
```

🔁 Method Overloading

🇮🇳 मराठी:

एकाच class मध्ये **same method** नाव,
पण **different parameters** असतील तर त्याला method overloading म्हणतात.

🇬🇧 English:

When multiple methods in the same class have the **same name but different parameters**,
it is called method overloading.

📦 Call by Value (Java)

🇮🇳 मराठी:

Java मध्ये **call by value** असते.
म्हणजे variable ची copy pass होते, original value change होत नाही.

🇬🇧 English:

Java follows **call by value**.
A copy of the variable is passed, so the original value does not change.

🇮🇳 Method चे Access Modifiers

🇮🇳 मराठी:

- public
- private
- protected
- default

🇬🇧 English:

Method access modifiers:

- public
- private
- protected
- default

🎯 Interview Important Points

🇮🇳 मराठी:

- Java मध्ये function ला method म्हणतात
- Method code reusability देतो
- Java call by value आहे
- main() ही static method आहे

🇬🇧 English:

- Functions are called methods in Java
- Methods provide code reusability
- Java follows call by value
- main() is a static method

✅Function (Method) Summary

🇮🇳 मराठी:

- Method = function in Java
- Method program structured बनवतो
- OOP चा important part आहे

🇬🇧 English:

- Method is a function in Java

- Methods structure programs
- Important part of OOP

Colletions

☐ Collections in Java

☐ Collection म्हणजे काय?

☐ मराठी:

Collection म्हणजे **multiple objects** एकत्र **store** करण्यासाठी वापरली जाणारी **framework**. Java मध्ये Collection Framework मुळे:

- Data dynamic पद्धतीने store होतो
- Array पेक्षा flexible असतो
- Ready-made classes आणि methods मिळतात

☐ English:

A **collection** is a **framework used to store and manipulate multiple objects**.
The Java Collection Framework:

- Stores data dynamically
- Is more flexible than arrays
- Provides ready-made classes and methods

☐ Collection Framework म्हणजे काय?

☐ मराठी:

Java Collection Framework म्हणजे:

- Interfaces
- Classes
- Algorithms

यांचा एक **standard architecture** आहे.

🇬🇧 English:

The Java Collection Framework is a **standard architecture** consisting of:

- Interfaces
- Classes
- Algorithms

🧱 Collection Framework Hierarchy

🇮🇳 मराठी:

मुख्य interfaces:

- Iterable
- Collection
- List
- Set
- Queue

Map हा Collection चा भाग नाही.

🔹 English:

Main interfaces:

- Iterable
- Collection
- List
- Set
- Queue

Map is **not** a part of the Collection interface.

🔹 List Interface

🔹 मराठी:

List:

- Duplicate elements allow करतो
- Insertion order maintain करतो

Common classes:

- ArrayList
- LinkedList
- Vector

Example:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
```

🔹 English:

List:

- Allows duplicate elements
- Maintains insertion order

Common classes:

- ArrayList
- LinkedList
- Vector

Example:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
```

🔹 Set Interface

🔹 मराठी:

Set:

- Duplicate elements allow करत नाही
- Order guarantee नसते

Common classes:

- HashSet

- LinkedHashSet
- TreeSet

🔤 English:

Set:

- Does not allow duplicate elements
- Order is not guaranteed

Common classes:

- HashSet
- LinkedHashSet
- TreeSet

🔤 Queue Interface

🔤 मराठी:

Queue:

- FIFO (First In First Out) follow करतो

Common classes:

- PriorityQueue
- ArrayDeque

🔤 English:

Queue:

- Follows FIFO (First In First Out)

Common classes:

- PriorityQueue
- ArrayDeque

🗺️ Map Interface

🔤 मराठी:

Map:

- Data **key-value pair** मध्ये store करतो
- Duplicate keys allow नाही

Common classes:

- HashMap
- LinkedHashMap
- TreeMap
- Hashtable

🔤 English:

Map:

- Stores data in **key-value pairs**
- Does not allow duplicate keys

Common classes:

- HashMap
- LinkedHashMap
- TreeMap
- Hashtable

? ArrayList vs LinkedList

? मराठी:

ArrayList	LinkedList
Dynamic array	Doubly linked list
Fast random access	Slow random access
Slow insertion/deletion	Fast insertion/deletion

? English:

ArrayList	LinkedList
Dynamic array	Doubly linked list
Fast random access	Slow random access
Slow insertion/deletion	Fast insertion/deletion

? Iterator

🇮🇳 मराठी:

Iterator वापरून collection मधील elements traverse करता येतात.

```
Iterator it = list.iterator();
while(it.hasNext()) {
    System.out.println(it.next());
}
```

🇬🇧 English:

An iterator is used to traverse elements in a collection.

```
Iterator it = list.iterator();
while(it.hasNext()) {
    System.out.println(it.next());
}
```

🇮🇳 Collections vs Collection

🇮🇳 मराठी:

- Collection → Interface
- Collections → Utility class

🇬🇧 English:

- Collection → Interface
- Collections → Utility class

✅ Advantages of Collection Framework

✅ मराठी:

- Ready-made data structures
- Performance improvement
- Code reuse

✅ English:

- Ready-made data structures
- Improved performance
- Code reuse

✅ Interview Important Points

✅ मराठी:

- Map Collection interface चा भाग नाही
- ArrayList thread-safe नाही
- Vector thread-safe आहे

✅ English:

- Map is not part of the Collection interface
- ArrayList is not thread-safe

- Vector is thread-safe

✅Collections Summary

🇮🇳 मराठी:

- Collections dynamic data store करतात
- List, Set, Queue, Map मुख्य parts आहेत
- Interview मध्ये खूप महत्वाचा topic

🇬🇧 English:

- Collections store data dynamically
- List, Set, Queue, Map are main parts
- Very important interview topic

ArrayList in Java (Deep Explanation)

ArrayList in Java (Deep Explanation)

ArrayList म्हणजे काय?

मराठी:

ArrayList ही Java मधील एक **List interface implement** करणारी **class** आहे.
ArrayList मध्ये:

- Duplicate elements allow असतात
- Insertion order maintain केला जातो
- Size dynamic असतो (आपोआप वाढतो / कमी होतो)

ArrayList internally **dynamic array** वापरतो.

English:

ArrayList is a class in Java that **implements the List interface**.
ArrayList:

- Allows duplicate elements
- Maintains insertion order
- Has dynamic size (automatically grows/shrinks)

Internally, ArrayList uses a **dynamic array**.

ArrayList का वापरतात?

मराठी:

- Data ची संख्या आधी माहीत नसेल तेव्हा
- Fast data access हवा असेल तेव्हा
- List type data store करायचा असेल तेव्हा

🇬🇧 English:

ArrayList is used:

- When the number of elements is not known
- When fast data access is required
- When list-type data needs to be stored

🧠 ArrayList कसा declare करतात?

🇮🇳 मराठी:

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

🇬🇧 English:

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

🧠 ArrayList मध्ये data add करणे

🇮🇳 मराठी:

```
ArrayList<String> names = new ArrayList<>();
names.add("Amit");
names.add("Rahul");
names.add("Amit"); // duplicate allowed
```

🔤 English:

```
ArrayList<String> names = new ArrayList<>();
names.add("Amit");
names.add("Rahul");
names.add("Amit"); // duplicate allowed
```

🔤 ArrayList मधून data access करणे

🔤 मराठी:

ArrayList मध्ये **index** वापरून **element** मिळतो.

```
System.out.println(names.get(0));
```

🔤 English:

Elements in ArrayList are accessed using **index**.

```
System.out.println(names.get(0));
```

🔤 ArrayList मधून data remove करणे

🔤 मराठी:

```
names.remove(1);         // index वरून
names.remove("Amit");  // object वरून
```

🔤 English:

```
names.remove(1);         // by index
names.remove("Amit");  // by object
```

✅ ArrayList चे Important Methods

✅ मराठी:

Method	काम
add()	element add करतो
get()	element fetch करतो
remove()	element delete करतो
size()	size देतो
contains()	element आहे का check
clear()	सगळा data delete

✅ English:

Method	Purpose
add()	Adds element
get()	Fetches element
remove()	Removes element
size()	Returns size
contains()	Checks element
clear()	Removes all elements

✅ ArrayList traversal (loop)

✅ मराठी:

```
for(String s : names) {
```

```
    System.out.println(s);
}
```

🔹 English:

```
for(String s : names) {
    System.out.println(s);
}
```

🔹 ArrayList vs Array

🔹 मराठी:

Array	ArrayList
Fixed size	Dynamic size
Primitive + object	Only objects
Fast	Slightly slower

🔹 English:

Array	ArrayList
Fixed size	Dynamic size
Stores primitives & objects	Stores only objects
Faster	Slightly slower

🔹 ArrayList internally कसा काम करतो? (Interview)

🔹 मराठी:

- Default capacity = **10**

Size full झाला की:

```
newCapacity = oldCapacity * 1.5
```

-
- जुना array copy होतो आणि नवीन तयार होतो

🇬🇧 English:

- Default capacity = **10**

When capacity is full:

```
newCapacity = oldCapacity * 1.5
```

-
- Old array is copied into a new array

🤔 ArrayList thread-safe आहे का?

🇮🇳 मराठी:

❌ArrayList thread-safe नाही
✔️Synchronization manually करावी लागते

```
Collections.synchronizedList(list);
```

🇬🇧 English:

❌ArrayList is not thread-safe
✔️Must be synchronized manually

```
Collections.synchronizedList(list);
```

🎯 Interview Important Points (Must Remember)

🇮🇳 मराठी:

- ArrayList duplicate allow करतो
- Insertion order maintain करतो
- Random access fast असतो
- Thread-safe नाही

🇬🇧 English:

- ArrayList allows duplicates
- Maintains insertion order
- Fast random access
- Not thread-safe

✅ ArrayList Summary

🇮🇳 मराठी:

- ArrayList dynamic list आहे
- Frequent read operations साठी best
- Collections मधील सवात जास्त वापरली जाणारी class

🇬🇧 English:

- ArrayList is a dynamic list

- Best for frequent read operations
- Most commonly used collection class

🔍 HashMap in Java (Deep Explanation)

🔥 HashMap in Java (Deep Explanation)

🔹 HashMap म्हणजे काय?

🔹 मराठी:

HashMap ही Java मधील एक **Map interface implement** करणारी **class** आहे.
HashMap मध्ये data **key–value pair** मध्ये store होतो.

HashMap चे features:

- Duplicate **keys allow** नाहीत
- Duplicate **values allow** आहेत
- Insertion order maintain होत नाही
- One `null` key allow आहे
- Multiple `null` values allow आहेत

🔹 English:

HashMap is a class in Java that **implements the Map interface**.
It stores data in **key–value pairs**.

Features of HashMap:

- Does not allow duplicate keys
- Allows duplicate values
- Does not maintain insertion order
- Allows one `null` key

- Allows multiple `null` values

🔹 HashMap का वापरतात?

🔹 मराठी:

- Fast searching, insertion, deletion साठी
- Key वापरून data लगेच मिळवण्यासाठी
- Large amount of data handle करण्यासाठी

🔹 English:

HashMap is used:

- For fast searching, insertion, and deletion
- To retrieve data quickly using keys
- To handle large amounts of data efficiently

🔹 HashMap कसा declare करतात?

🔹 मराठी:

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

🔹 English:

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

🔹 HashMap मध्ये data add करणे

🇮🇳 मराठी:

```
map.put(1, "Amit");
map.put(2, "Rahul");
map.put(3, "Neha");
map.put(1, "Suresh"); // key duplicate → value replace होते
```

🌍 English:

```
map.put(1, "Amit");
map.put(2, "Rahul");
map.put(3, "Neha");
map.put(1, "Suresh"); // duplicate key → value replaced
```

🔹 HashMap मधून data मिळवणे

🇮🇳 मराठी:

Key वापरून value मिळवतात.

```
System.out.println(map.get(2));
```

🌍 English:

Values are retrieved using keys.

```
System.out.println(map.get(2));
```

🔹 HashMap मधून data remove करणे

🇮🇳 मराठी:

```
map.remove(3);
```

🔹 English:

```
map.remove(3);
```

🔹 HashMap चे Important Methods

🔹 मराठी:

Method	काम
put()	data add करतो
get()	value fetch करतो
remove()	data delete
containsKey()	key आहे का check
containsValue()	value आहे का check
size()	total entries
isEmpty()	map empty आहे का

🔹 English:

Method	Purpose
put()	Adds data
get()	Fetches value
remove()	Deletes entry
containsKey()	Checks key
containsValue()	Checks value
size()	Total entries

isEmpty()	Checks empty

🔁 HashMap traversal (Iteration)

🇮🇳 मराठी:

```
for(Map.Entry<Integer, String> entry : map.entrySet()) {
    System.out.println(entry.getKey() + " " + entry.getValue());
}
```

🇬🇧 English:

```
for(Map.Entry<Integer, String> entry : map.entrySet()) {
    System.out.println(entry.getKey() + " " + entry.getValue());
}
```

🧠 HashMap internally कसा काम करतो? (Very Important)

🇮🇳 मराठी:

- HashMap internally **array + linked list / tree** वापरतो
- Key चा **hashCode()** काढला जातो
- Hash code वापरून **bucket index** ठरतो
- Collision झाला तर:
    - LinkedList वापरली जाते
    - Java 8 नंतर large collision साठी **Red-Black Tree** वापरतो

Default:

- Initial capacity = **16**

- Load factor = **0.75**

🇬🇧 English:

- HashMap internally uses **array + linked list / tree**
- `hashCode()` of the key is calculated
- Hash code determines the **bucket index**
- If collision occurs:
  - LinkedList is used
  - From Java 8 onwards, **Red-Black Tree** is used for large collisions

Defaults:

- Initial capacity = **16**
- Load factor = **0.75**

🆚 HashMap vs Hashtable

🇮🇳 मराठी:

HashMap	Hashtable
Not thread-safe	Thread-safe
Allows null key/value	No null key/value
Faster	Slower

🇬🇧 English:

HashMap	Hashtable
Not thread-safe	Thread-safe
Allows null key/value	No null key/value
Faster	Slower

🔹 HashMap thread-safe कसा करायचा?

🔹 मराठी:

```
Map map = Collections.synchronizedMap(new HashMap<>());
```

🔹 English:

```
Map map = Collections.synchronizedMap(new HashMap<>());
```

🔹 Interview Important Points (Must Remember)

🔹 मराठी:

- HashMap insertion order maintain करत नाही
- One null key allow आहे
- hashCode() आणि equals() महत्वाचे
- Java 8 मध्ये tree structure वापरतो

🔹 English:

- HashMap does not maintain insertion order
- Allows one null key

- hashCode() and equals() are important
- Uses tree structure in Java 8

✅HashMap Summary

🇮🇳 मराठी:

- HashMap fast performance देतो
- Key–value data store करण्यासाठी best
- Interview मध्ये खूप विचारला जाणारा topic

🇬🇧 English:

- HashMap provides fast performance
- Best for storing key–value data
- Very frequently asked interview topic

□ Set in Java (Deep Explanation)

❓ Set in Java (Deep Explanation)

❓ Set म्हणजे काय?

❓ मराठी:

Set ही Java मधील एक **Collection interface** आहे जी:

- **Duplicate elements allow** करत नाही
- Elements unique ठेवते
- Insertion order ची guarantee देत नाही (implementation वर depend असते)

Set चा वापर तेव्हा करतात जेव्हा **unique data** हवा असतो.

❓ English:

Set is a **Collection interface** in Java that:

- Does **not allow duplicate elements**
- Stores unique elements
- Does not guarantee insertion order (depends on implementation)

Set is used when **unique data** is required.

❓ Set Interface implement करणाऱ्या Classes

❓ मराठी:

Set interface implement करणाऱ्या मुख्य classes:

1. HashSet
2. LinkedHashSet
3. TreeSet

🇬🇧 English:

Main classes that implement the Set interface:

1. HashSet
2. LinkedHashSet
3. TreeSet

1️⃣ HashSet

🇮🇳 मराठी:

HashSet:

- Duplicate elements allow नाही
- Insertion order maintain होत नाही
- One `null` element allow आहे
- Fast performance देते

Internally **HashMap** वापरतो.

Example:

```
HashSet<Integer> set = new HashSet<>();
set.add(10);
set.add(20);
```

```
set.add(10); // duplicate ignore होईल
```

🔤 English:

HashSet:

- Does not allow duplicate elements
- Does not maintain insertion order
- Allows one `null` element
- Provides fast performance

Internally uses a **HashMap**.

Example:

```
HashSet<Integer> set = new HashSet<>();
set.add(10);
set.add(20);
set.add(10); // duplicate ignored
```

2️⃣ LinkedHashSet

🔤 मराठी:

LinkedHashSet:

- Duplicate elements allow नाही
- **Insertion order maintain** करतो
- One `null` element allow आहे
- HashSet पेक्षा थोडा slow

Example:

```
LinkedHashSet<String> set = new LinkedHashSet<>();
set.add("A");
set.add("B");
set.add("A");
```

🔹 English:

LinkedHashSet:

- Does not allow duplicates
- **Maintains insertion order**
- Allows one `null` element
- Slightly slower than HashSet

Example:

```
LinkedHashSet<String> set = new LinkedHashSet<>();
set.add("A");
set.add("B");
set.add("A");
```

3️⃣ TreeSet

🔹 मराठी:

TreeSet:

- Duplicate elements allow नाही
- Elements **sorted order** मध्ये store होतात
- `null` element allow नाही

- Slower than HashSet (tree structure वापरतो)

Internally **Red-Black Tree** वापरतो.

Example:

```
TreeSet<Integer> set = new TreeSet<>();
set.add(30);
set.add(10);
set.add(20);
```

🌍 English:

TreeSet:

- Does not allow duplicate elements
- Stores elements in **sorted order**
- Does not allow `null` elements
- Slower than HashSet (uses tree structure)

Internally uses a **Red-Black Tree**.

Example:

```
TreeSet<Integer> set = new TreeSet<>();
set.add(30);
set.add(10);
set.add(20);
```

🆚 HashSet vs LinkedHashSet vs TreeSet

🇮🇳 मराठी:

Feature	HashSet	LinkedHashSet	TreeSet

Duplicate	✗	✗	✗
Order	No	Insertion	Sorted
Null	1 allowed	1 allowed	Not allowed
Speed	Fastest	Medium	Slow

🇬🇧 English:

Feature	HashSet	LinkedHashSet	TreeSet
Duplicates	✗	✗	✗
Order	No order	Insertion order	Sorted
Null	One allowed	One allowed	Not allowed
Speed	Fastest	Medium	Slow

🔁 Set Traversal (Iteration)

🇮🇳 मराठी:

Set iterate करण्यासाठी **for-each loop** किंवा **Iterator** वापरतात.

```
for(Integer i : set) {
    System.out.println(i);
}
```

🇬🇧 English:

To iterate over a Set, use **for-each loop** or **Iterator**.

```
for(Integer i : set) {
    System.out.println(i);
}
```

🔹 Set कधी वापरायचा? (Real Use)

🔹 मराठी:

- Duplicate values remove करायच्या असतील
- Unique IDs / usernames store करायचे असतील
- Sorted unique data हवा असेल (TreeSet)

🔹 English:

Use Set when:

- Duplicate values must be removed
- Unique IDs / usernames are needed
- Sorted unique data is required (TreeSet)

🔹 Interview Important Points

🔹 मराठी:

- Set duplicate elements allow करत नाही
- HashSet internally HashMap वापरतो
- TreeSet sorted order देतो
- TreeSet null allow करत नाही

🔹 English:

- Set does not allow duplicates
- HashSet internally uses HashMap
- TreeSet provides sorted order
- TreeSet does not allow null

✅Set Summary

🇮🇳 मराठी:

- Set unique data साठी वापरतात
- HashSet fastest आहे
- LinkedHashSet order maintain करतो
- TreeSet sorted data देतो

🇬🇧 English:

- Set is used for unique data
- HashSet is the fastest
- LinkedHashSet maintains order
- TreeSet provides sorted data

□ Comparable vs Comparator in Java

? Comparable vs Comparator in Java

? Comparable म्हणजे काय?

? मराठी:

Comparable ही Java मधील एक **interface** आहे जी **natural sorting order define** करण्यासाठी वापरतात.
Class स्वतः सांगतो की objects कसे compare व्हावेत.

- `compareTo()` method वापरतो
- Class च्या आत implement करतात
- एकाच **sorting logic** ला support

Syntax:

```
class Student implements Comparable<Student> {
    public int compareTo(Student s) {
        // logic
    }
}
```

? English:

Comparable is an **interface** in Java used to define the **natural ordering** of objects.
The class itself defines how objects should be compared.

- Uses `compareTo()` method
- Implemented inside the class
- Supports **only one sorting logic**

Syntax:

```
class Student implements Comparable<Student> {
    public int compareTo(Student s) {
        // logic
    }
}
```

□ Comparable Example

□ मराठी:

खाली Student objects **id** नुसार **sort** केले आहेत.

```
class Student implements Comparable<Student> {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int compareTo(Student s) {
        return this.id - s.id;
    }
}
```

□ English:

Below example sorts Student objects **by id**.

```
class Student implements Comparable<Student> {
    int id;
    String name;

    Student(int id, String name) {
```

```
        this.id = id;
        this.name = name;
    }

    public int compareTo(Student s) {
        return this.id - s.id;
    }
}
```

🔹 Comparator म्हणजे काय?

🔹 मराठी:

Comparator ही Java मधील एक **interface** आहे जी **custom sorting logic** देण्यासाठी वापरतात.
Sorting logic **class** च्या बाहेर define केली जाते.

- `compare()` method वापरतो
- Multiple sorting logics शक्य
- Class modify करायची गरज नाही

Syntax:

```
class NameComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2) {
        return s1.name.compareTo(s2.name);
    }
}
```

🔹 English:

Comparator is an **interface** used to define **custom sorting logic**.
The sorting logic is defined **outside the class**.

- Uses `compare()` method

- Supports multiple sorting logics
- No need to modify the original class

Syntax:

```
class NameComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2) {
        return s1.name.compareTo(s2.name);
    }
}
```

□ Comparator Example

□ मराठी:

Student objects **name** नुसार **sort** करणे.

```
Collections.sort(list, new NameComparator());
```

□ English:

Sorting Student objects **by name**.

```
Collections.sort(list, new NameComparator());
```

□ Comparable vs Comparator (Main Difference)

□ मराठी:

Comparable	Comparator
`compareTo()`	`compare()`
Class च्या आत	Class च्या बाहेर

One sorting logic	Multiple sorting logics
Original class modify	No modification needed

🇬🇧 English:

Comparable	Comparator
`compareTo()`	`compare()`
Inside the class	Outside the class
One sorting logic	Multiple sorting logics
Class must be modified	No modification needed

🔹 Collections.sort() सोबत वापर

🇮🇳 मराठी:

- Comparable → `Collections.sort(list)`
- Comparator → `Collections.sort(list, comparator)`

🇬🇧 English:

- Comparable → `Collections.sort(list)`
- Comparator → `Collections.sort(list, comparator)`

🔹 Interview Important Points ⭐

🇮🇳 मराठी:

- Comparable natural sorting देतो
- Comparator custom sorting देतो
- Comparator flexible आहे
- TreeSet / TreeMap sorting साठी वापरले जातात

🇬🇧 English:

- Comparable provides natural ordering
- Comparator provides custom ordering
- Comparator is more flexible
- Used with TreeSet / TreeMap for sorting

🌍 Real-Time Use Case

🇮🇳 मराठी:

Employee objects:

- Salary नुसार sort → Comparator
- ID नुसार default sort → Comparable

🇬🇧 English:

Employee objects:

- Sort by salary → Comparator

- Default sort by ID → Comparable

✅Comparable vs Comparator Summary

🇮🇳 मराठी:

- Comparable = natural order
- Comparator = custom order
- Interview मध्ये खूप विचारला जाणारा topic

🇬🇧 English:

- Comparable = natural order
- Comparator = custom order
- Very frequently asked interview topic

Exception

☐ Exception Handling in Java

☐ Exception म्हणजे काय?

☐ मराठी:

Exception म्हणजे program run होत असताना येणारी **runtime error situation**.
Exception आली तर program अचानक थांबू शकतो, म्हणून ती **handle** करणं गरजेचं असतं.

उदा:

- 0 ने division
- Array index out of range
- Null object access

☐ English:

An **exception** is a **runtime error** that occurs during program execution.
If not handled, it can terminate the program abnormally, so it must be handled properly.

Examples:

- Division by zero
- Array index out of range
- Accessing a null object

☐ Exception Handling का गरजेचं आहे?

☐ मराठी:

- Program crash होऊ नये म्हणून
- Normal execution चालू ठेवण्यासाठी
- Error proper message देण्यासाठी

🇬🇧 English:

Exception handling is required to:

- Prevent program crashes
- Maintain normal program flow
- Provide proper error messages

🇮🇳 Java मध्ये Exception Handling Keywords

🇮🇳 मराठी:

Java मध्ये exception handle करण्यासाठी खालील keywords वापरतात:

- `try`
- `catch`
- `finally`
- `throw`
- `throws`

🇬🇧 English:

Java provides the following keywords for exception handling:

- `try`
- `catch`
- `finally`
- `throw`
- `throws`

🔹 try–catch Block

🔹 मराठी:

`try` block मध्ये **error** येऊ शकणारा **code** लिहितात.
`catch` block मध्ये **exception handle** करतात.

```
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero");
}
```

🔹 English:

The `try` block contains code that may cause an exception.
The `catch` block handles the exception.

```
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero");
}
```

🔹 Multiple catch Blocks

🇮🇳 मराठी:

एका `try` block साठी **multiple catch blocks** वापरू शकतो.

```
try {
    int arr[] = new int[5];
    arr[10] = 20;
} catch (ArithmeticException e) {
    System.out.println("Arithmetic error");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Array index error");
}
```

🇬🇧 English:

Multiple `catch` blocks can be used with a single `try` block.

```
try {
    int arr[] = new int[5];
    arr[10] = 20;
} catch (ArithmeticException e) {
    System.out.println("Arithmetic error");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Array index error");
}
```

🔹 finally Block

🇮🇳 मराठी:

`finally` block **exception** आली किंवा नाही आली तरी **execute** होतो.
Database close, file close साठी वापरतात.

```
finally {
    System.out.println("Finally block executed");
}
```

🔹 English:

The `finally` block **always executes**, whether an exception occurs or not.
It is used to close resources like files or database connections.

```
finally {
    System.out.println("Finally block executed");
}
```

🔹 throw Keyword

🔹 मराठी:

`throw` keyword वापरून **manually exception create** करता येते.

```
throw new ArithmeticException("Invalid operation");
```

🔹 English:

The `throw` keyword is used to **manually create an exception**.

```
throw new ArithmeticException("Invalid operation");
```

🔹 throws Keyword

🔹 मराठी:

`throws` keyword वापरून **method exception** पुढे **pass** करते.

```
void readFile() throws IOException {
```

```
}
```

🇬🇧 English:

The `throws` keyword declares that a method **may pass the exception to the caller**.

```
void readFile() throws IOException {
}
```

🔹 Types of Exceptions

🇮🇳 मराठी:

Java मध्ये exception चे 2 मुख्य प्रकार आहेत:

1. Checked Exception
2. Unchecked Exception

🇬🇧 English:

Java has two main types of exceptions:

1. Checked exceptions
2. Unchecked exceptions

🔹 Checked Exception

🇮🇳 मराठी:

Checked exception **compile time** ला **check** होतो.
Handle करणं compulsory असतं.

उदा:

- IOException
- SQLException

🔹 English:

Checked exceptions are **checked at compile time**.
Handling them is mandatory.

Examples:

- IOException
- SQLException

🔹 Unchecked Exception

🔹 मराठी:

Unchecked exception **runtime** ला येतो.
Handle करणं compulsory नाही.

उदा:

- ArithmeticException
- NullPointerException

🔹 English:

Unchecked exceptions occur at **runtime**.
Handling them is not mandatory.

Examples:

- ArithmeticException
- NullPointerException

🔹 Custom Exception

🔹 मराठी:

आपण स्वतःची exception class तयार करू शकतो.

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

🔹 English:

We can create our own custom exception class.

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

🔹 Interview Important Points ⭐

🔹 मराठी:

- Exception runtime error आहे
- finally block always execute होतो

- Checked exception handle करणे compulsory
- throw vs throws difference महत्वाचा

🇬🇧 English:

- Exceptions are runtime errors
- finally block always executes
- Checked exceptions must be handled
- Difference between throw and throws is important

✅Exception Handling Summary

🇮🇳 मराठी:

- Exception handling program safe बनवते
- try–catch–finally मुख्य blocks आहेत
- Interview मध्ये खूप महत्वाचा topic

🇬🇧 English:

- Exception handling makes programs safe
- try–catch–finally are core blocks
- Very important interview topic

Multithreading in Java

🧵 Multithreading in Java

🔹 Thread म्हणजे काय?

🇮🇳 मराठी:

Thread म्हणजे program मधील एक **independent path of execution**.
एकाच program मध्ये एकापेक्षा जास्त threads असतील तर त्याला **Multithreading** म्हणतात. उदा:

- Music app → गाणं play + download
- Browser → page load + scroll

🇬🇧 English:

A **thread** is an **independent path of execution** within a program.
When multiple threads run simultaneously in a program, it is called **multithreading**.

Examples:

- Music app → play music + download
- Browser → page loading + scrolling

🔹 Multithreading का वापरतात?

🇮🇳 मराठी:

- CPU utilization वाढवण्यासाठी
- Program fast चालवण्यासाठी

- Multiple tasks एकाच वेळी करण्यासाठी

🇬🇧 English:

Multithreading is used to:

- Improve CPU utilization
- Increase program performance
- Perform multiple tasks simultaneously

🧵 Thread तयार करण्याचे माग

🇮🇳 मराठी:

Java मध्ये thread तयार करण्याचे **2 main ways** आहेत:

1. Thread class extend करून
2. Runnable interface implement करून

🇬🇧 English:

There are **two main ways** to create a thread in Java:

1. By extending the Thread class
2. By implementing the Runnable interface

1️⃣ Thread class extend करून

🇮🇳 मराठी:

Thread class extend करून `run()` method override करतात.
`start()` method call केल्यावर thread execute होतो.

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

class Test {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

🇬🇧 English:

By extending the Thread class, we override the `run()` method.
Calling `start()` starts a new thread.

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

class Test {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

2️⃣ Runnable interface implement करून

🇮🇳 मराठी:

Runnable interface implement करून thread तयार करणे **best practice** आहे.
कारण Java multiple inheritance allow करत नाही.

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread running");
    }
}

class Test {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

🇬🇧 English:

Implementing the Runnable interface is the **best practice**.
Java does not support multiple inheritance.

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread running");
    }
}

class Test {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

🔁 Thread Life Cycle

🇮🇳 मराठी:

Thread चे states:

1. New
2. Runnable
3. Running
4. Waiting / Blocked
5. Dead

🇬🇧 English:

Thread states:

1. New
2. Runnable
3. Running
4. Waiting / Blocked
5. Dead

🔁 Important Thread Methods

🇮🇳 मराठी:

Method	काम
start()	Thread सुरू

	करतो
run()	Thread logic
sleep()	Thread थांबवतो
join()	दुसऱ्या thread ची वाट पाहतो
getName()	Thread नाव
setPriority()	Priority set

🇬🇧 English:

Method	Purpose
start()	Starts thread
run()	Thread logic
sleep()	Pauses thread
join()	Waits for another thread
getName()	Thread name
setPriority()	Sets priority

🔢 Thread Priority

🇮🇳 मराठी:

Thread priority 1 ते 10 दरम्यान असते:

- MIN_PRIORITY = 1
- NORM_PRIORITY = 5
- MAX_PRIORITY = 10

- **English:**

Thread priority ranges from 1 to 10:

- MIN_PRIORITY = 1
- NORM_PRIORITY = 5
- MAX_PRIORITY = 10

🔒 Synchronization

🇮🇳 मराठी:

Synchronization वापरून **shared resource** सुरक्षित करता येतो.
एकाच वेळी एकच thread resource वापरू शकतो.

```
synchronized void print() {
}
```

🇬🇧 English:

Synchronization is used to **protect shared resources**.
Only one thread can access the resource at a time.

```
synchronized void print() {
}
```

🔒 Deadlock म्हणजे काय?

🇮🇳 मराठी:

Deadlock म्हणजे:

- दोन threads एकमेकांच्या resource साठी वाट पाहतात
- Program अडकतो

🇬🇧 English:

Deadlock occurs when:

- Two threads wait for each other’s resources
- Program gets stuck

🔁 Thread vs Process

🇮🇳 मराठी:

Thread	Process
Lightweight	Heavyweight
Same memory share	Separate memory
Fast	Slow

🇬🇧 English:

Thread	Process
Lightweight	Heavyweight
Shares memory	Separate memory
Faster	Slower

🎯 Interview Important Points ⭐

🇮🇳 मराठी:

- start() new thread तयार करतो
- run() direct call केल्यास new thread नाही
- Runnable better than Thread
- Synchronization race condition टाळतो

🇬🇧 English:

- start() creates a new thread
- Calling run() directly does not create a new thread
- Runnable is better than Thread
- Synchronization avoids race conditions

✅Multithreading Summary

🇮🇳 मराठी:

- Multithreading performance वाढवते
- Thread class किंवा Runnable वापरतात
- Interview मध्ये खूप महत्वाचा topic

🇬🇧 English:

- Multithreading improves performance
- Use Thread or Runnable
- Very important interview topic

Wrapper Classes in Java

🔹 Wrapper Classes in Java

🔹 Wrapper Class म्हणजे काय?

🔹 मराठी:

Wrapper Class म्हणजे **primitive data type** ला **object** मध्ये **convert** करणारी **class**.
Java मध्ये collections (ArrayList, HashMap) फक्त **objects** accept करतात,
म्हणून primitive data साठी wrapper classes वापरतात.

Primitive → Object

उदा:

- int → Integer
- char → Character

🔹 English:

A **wrapper class** is a class that **converts a primitive data type into an object**.
Java collections (ArrayList, HashMap) work only with **objects**,
so wrapper classes are used for primitive data types.

Primitive → Object

Example:

- int → Integer
- char → Character

🔹 Primitive Types आणि Wrapper Classes

🔹 मराठी:

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

🔹 English:

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

🔹 Wrapper Class का वापरतात?

🔹 मराठी:

- Collections मध्ये primitive data store करण्यासाठी

- Object-oriented features वापरण्यासाठी
- Utility methods (parseInt, valueOf) वापरण्यासाठी

🔤 English:

Wrapper classes are used to:

- Store primitive data in collections
- Use object-oriented features
- Use utility methods like parseInt, valueOf

🔤 Boxing म्हणजे काय?

🔤 मराठी:

Boxing म्हणजे **primitive value** ला **wrapper object** मध्ये **convert** करणे.

```
int a = 10;
Integer i = Integer.valueOf(a);
```

🔤 English:

Boxing is the process of **converting a primitive value into a wrapper object**.

```
int a = 10;
Integer i = Integer.valueOf(a);
```

🔤 Unboxing म्हणजे काय?

🔤 मराठी:

Unboxing म्हणजे **wrapper object** ला **primitive value** मध्ये **convert** करणे.

```
Integer i = 20;
int a = i.intValue();
```

🇬🇧 English:

Unboxing is the process of **converting a wrapper object into a primitive value**.

```
Integer i = 20;
int a = i.intValue();
```

🔹 Autoboxing

🇮🇳 मराठी:

Autoboxing मध्ये Java automatically primitive → object convert करते.

```
int a = 10;
Integer i = a; // autoboxing
```

🇬🇧 English:

In **autoboxing**, Java automatically converts primitive values into objects.

```
int a = 10;
Integer i = a; // autoboxing
```

🔹 Auto-unboxing

🇮🇳 मराठी:

Java automatically object → primitive convert करते.

```
Integer i = 20;
int a = i; // auto-unboxing
```

🇬🇧 English:

Java automatically converts wrapper objects into primitive values.

```
Integer i = 20;
int a = i; // auto-unboxing
```

🔧 Important Wrapper Class Methods

🇮🇳 मराठी:

Method	काम
valueOf()	primitive → object
parseInt()	String → primitive
toString()	value → String
compareTo()	comparison

🇬🇧 English:

Method	Purpose
valueOf()	primitive → object
parseInt()	String → primitive
toString()	value → String
compareTo()	comparison

🧪 Example: String to int conversion

🔹 मराठी:

```
String s = "123";
int a = Integer.parseInt(s);
```

🔹 **English:**

```
String s = "123";
int a = Integer.parseInt(s);
```

🔹 **Wrapper Class Immutable** का आहे**?**

🔹 मराठी:

Wrapper classes **immutable** आहेत म्हणजे
एकदा object तयार झाला की त्याची value change होत नाही.
New value साठी नवीन object तयार होतो.

🔹 **English:**

Wrapper classes are **immutable**, meaning
once an object is created, its value cannot be changed.
A new object is created for a new value.

🔹 **Wrapper Classes vs Primitive Types**

🔹 मराठी:

Primitive	**Wrapper**
Fast	Slightly slower
Less memory	More memory
No methods	Many utility methods

🇬🇧 English:

Primitive	Wrapper
Faster	Slightly slower
Less memory	More memory
No methods	Many utility methods

🎯 Interview Important Points ⭐

🇮🇳 मराठी:

- Collections primitive support करत नाहीत
- Autoboxing Java 5 पासून आले
- Wrapper classes immutable आहेत
- Integer cache (-128 to 127) concept आहे

🇬🇧 English:

- Collections do not support primitives
- Autoboxing introduced in Java 5
- Wrapper classes are immutable
- Integer caching (-128 to 127) exists

✅Wrapper Class Summary

🇮🇳 मराठी:

- Wrapper class primitive ला object बनवते
- Boxing / Unboxing महत्वाचे concepts
- Collections साठी खूप आवश्यक

🇬🇧 English:

- Wrapper classes convert primitives to objects
- Boxing / Unboxing are key concepts
- Essential for collections

📁 File Handling in Java

📁 File Handling in Java

📌 File Handling म्हणजे काय?

🇮🇳 मराठी:

File Handling म्हणजे **file** मधील **data create, read, write, update** आणि **delete** करणे.
Java मध्ये file handling वापरून data **permanently store** करता येतो.

🌍 English:

File Handling is the process of **creating, reading, writing, updating, and deleting data in files**.
Java allows permanent data storage using file handling.

🤔 Java मध्ये File Handling का वापरतात?

🇮🇳 मराठी:

- Data permanently store करण्यासाठी
- Program बंद झाल्यावरही data सुरक्षित ठेवण्यासाठी
- Logs, reports, configuration files साठी

🌍 English:

File handling is used to:

- Store data permanently

- Preserve data after program termination
- Handle logs, reports, and configuration files

📂 Java File Handling Packages

🇮🇳 मराठी:

Java मध्ये file handling साठी:

- `java.io`
- `java.nio`

packages वापरतात.

🇬🇧 English:

Java provides file handling through:

- `java.io`
- `java.nio`

packages.

📄 File Class (`java.io.File`)

🇮🇳 मराठी:

`File` class वापरून:

- File create/delete करता येते

- File information मिळते (name, size, path)

```
File f = new File("test.txt");
f.createNewFile();
```

🇬🇧 English:

The `File` class is used to:

- Create/delete files
- Get file information (name, size, path)

```
File f = new File("test.txt");
f.createNewFile();
```

📝 File Writing (Writing Data into File)

🇮🇳 मराठी:

File मध्ये data लिहिण्यासाठी:

- FileWriter
- BufferedWriter
- PrintWriter

```
FileWriter fw = new FileWriter("test.txt");
fw.write("Hello Java");
fw.close();
```

🇬🇧 English:

To write data into a file, use:

- FileWriter
- BufferedWriter
- PrintWriter

```
FileWriter fw = new FileWriter("test.txt");
fw.write("Hello Java");
fw.close();
```

📖 File Reading (Reading Data from File)

🇮🇳 मराठी:

File मधील data वाचण्यासाठी:

- FileReader
- BufferedReader
- Scanner

```
BufferedReader br = new BufferedReader(new FileReader("test.txt"));
String line = br.readLine();
```

🇬🇧 English:

To read data from a file, use:

- FileReader
- BufferedReader
- Scanner

```
BufferedReader br = new BufferedReader(new FileReader("test.txt"));
String line = br.readLine();
```

🔹 try-with-resources (Java 7+)

🇮🇳 मराठी:

`try-with-resources` वापरल्यावर:

- File automatic close होते
- finally block ची गरज राहत नाही

```
try (FileReader fr = new FileReader("test.txt")) {
}
```

🇬🇧 English:

Using `try-with-resources`:

- Resources are closed automatically
- No need for finally block

```
try (FileReader fr = new FileReader("test.txt")) {
}
```

🔹 File Append Mode

🇮🇳 मराठी:

File append करण्यासाठी `true` pass करतात.

```
FileWriter fw = new FileWriter("test.txt", true);
```

🇬🇧 English:

To append data to a file, pass `true`.

```
FileWriter fw = new FileWriter("test.txt", true);
```

📝 Serialization

🇮🇳 मराठी:

Serialization म्हणजे object ला file मध्ये save करणे.

```
ObjectOutputStream oos =
 new ObjectOutputStream(new FileOutputStream("obj.txt"));
oos.writeObject(obj);
```

🇬🇧 English:

Serialization is the process of saving an object into a file.

```
ObjectOutputStream oos =
 new ObjectOutputStream(new FileOutputStream("obj.txt"));
oos.writeObject(obj);
```

📝 Deserialization

🇮🇳 मराठी:

File मधून object परत मिळवणे.

```
ObjectInputStream ois =
 new ObjectInputStream(new FileInputStream("obj.txt"));
```

🔹 English:

Reading an object back from a file.

```
ObjectInputStream ois =
 new ObjectInputStream(new FileInputStream("obj.txt"));
```

🔹 Checked Exceptions in File Handling

🔹 मराठी:

File handling मध्ये:

- IOException
- FileNotFoundException

handle कराव्या लागतात.

🔹 English:

File handling requires handling:

- IOException
- FileNotFoundException

🔹 Interview Important Points ⭐

🔹 मराठी:

- File class file content वाचत नाही

- try-with-resources best practice आहे
- Serialization साठी class Serializable implement करावी लागते

🇬🇧 English:

- File class does not read file content
- try-with-resources is best practice
- Class must implement Serializable for serialization

✅File Handling Summary

🇮🇳 मराठी:

- File handling permanent storage देते
- java.io package जास्त वापरला जातो
- Interview मध्ये खूप महत्वाचा topic

🇬🇧 English:

- File handling provides permanent storage
- java.io package is widely used
- Very important interview topic
