Real time interview Questions!!

@MrJavaDecode

Explain JVM and JIT ?

JVM (Java virtual machine):

The JVM is the engine that runs Java applications. It converts **compiled Java bytecode** (from .class files) into **machine code** at runtime so it can be executed on any platform. This is what enables Java's **"write once, run anywhere"** capability.

JIT (Just In Time Compiler):

The JIT is a part of the JVM that boosts performance by **compiling frequently used bytecode into native machine code at runtime**. This reduces the need for repeated interpretation and speeds up execution.

@MrJavaDecode

What is cloning and explain in brief ?

Cloning in Java means creating an **exact copy** of an object with the same values. It's done using the clone() method provided by the Object class.

There are two types of cloning:

- ✅ **Shallow Cloning**
  Creates a new object and copies **primitive fields** directly.
  For **reference fields**, it copies the reference (not the actual object), meaning changes in the original referenced object will affect the clone.
  Achieved by simply calling super.clone().
- ✅ **Deep Cloning**
  Copies not just the object, but also the objects referenced inside it (recursive copy).
  Each object is cloned individually, so changes in one don't affect the other.
  Requires custom implementation or use of serialization or external libraries.

@MrJavaDecode

Shallow cloning code example!

```
class Employee implements Cloneable {
    int id;
    String name;

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

    protected Object clone() throws
CloneNotSupportedException {
        return super.clone();
    }
}
```

@MrJavaDecode

What is default constructor ?

A default constructor is a no-argument constructor that is automatically provided by the compiler if no other constructor is defined in the class.

```
public class Car {
    String model;
    int year;

    // Default constructor
    public Car() {
        model = "Unknown";
        year = 0;
    }

    public void display() {
        System.out.println("Model: " + model + ", Year: " + year);
    }

    public static void main(String[] args) {
        Car c = new Car();   // Default constructor called
        c.display();
    }
}
```

@MrJavaDecode

What is a Wrapper Class in Java?

@MrJavaDecode

In Java, wrapper classes are used to wrap primitive data types (int, char, boolean, etc.) into objects.

Java provides a wrapper class for each primitive type:

int → Integer
char → Character
boolean → Boolean
double → Double
Etc.

Why Use Wrapper Classes?

- **Needed when objects are required instead of primitives (e.g., in collections like ArrayList, HashMap)**
- **For conversion between strings and numbers (Integer.parseInt("10"))**
- **To use utility methods (like Integer.MAX_VALUE, Character.isDigit())**

Autoboxing: **Automatically converting a primitive into a wrapper.**
Unboxing: **Automatically converting a wrapper back into a primitive.**

Why is String Immutable in Java?

In Java, a String is immutable, meaning once created, its value cannot be changed. Any modification creates a new String object.

Reasons for String Immutability:

Security

String Pooling

Thread Safety

Hashcode Consistency

Ex:

```
String str = "Java";
str.concat(" Developer");
System.out.println(str);  // Output: Java
```

@MrJavaDecode

@MrJavaDecode

Qtn. # Is it fine to modify collection while iterating ?

Modifying a collection while iterating over it in Java is generally not safe and can lead to a ConcurrentModificationException. This happens because most of Java's collection classes (like ArrayList, HashSet, etc.) are fail-fast — meaning they detect concurrent structural modification and fail immediately.

Unsafe: Modifying an ArrayList during iteration

```java
package mrjavadecode.practice;

//unsafe code will throw ConcurrentModificationException exception

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class UnsafeModification {

    public static void main(String[] args) {

        List<Integer> numbers = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

        for (Integer num : numbers) {
            if (num == 3) {
                numbers.remove(num); // Throws ConcurrentModificationException
            }
        }
    }

}
```

Problems | Javadoc | Declaration | Console

<terminated> UnsafeModification [Java Application] C:\Users\shash\Downloads\spring-tools-for-eclipse-4.31.0.RELEASE-e4.36.0-win32.win32.x86_64\sts-4.31.0.RELEASE\plu

```
Exception in thread "main" java.util.ConcurrentModificationException
        at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1095)
        at java.base/java.util.ArrayList$Itr.next(ArrayList.java:1049)
        at mrjavadecode/mrjavadecode.practice.UnsafeModification.main(UnsafeModification.java:13)
```

Safe by #1: Use an Iterator explicitly

@MrJavaDecode

```java
package mrjavadecode.practice;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SafeWithTemp {

    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>(Arrays
                .asList(1, 2, 3, 4, 5));
        List<Integer> toRemove = new ArrayList<>();

        for (Integer num : numbers) {
            if (num == 3) {
                toRemove.add(num);
            }
        }

        numbers.removeAll(toRemove);
        System.out.println(numbers); // Output:[1, 2, 4, 5]
    }

}
```

```
<terminated> SafeWithTemp
[1, 2, 4, 5]
```

Safe By #2 Use removeIf (Java 8+)

@MrJavaDecode

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class RemoveIfExample {
    public static void main(String[] args) {
        List<String> animal = new ArrayList<>(Arrays
                .asList("Cat", "Dog", "Lion"));
        animal.removeIf(fruit ->
        fruit.equals("Dog")); // ☑ Cleaner alternative
        System.out.println(animal); // Output: [Apple, Orange]
    }
}
```

Console

terminated> RemoveIfExample [Java Application] C:\Users\shash\Downloads\spring-tools-for-eclipse-4.31.0.RELEASE-e4.36.0-win32.win32.x86_64\sts-4.3

```
[Cat, Lion]
```

What is marker interface ?

A marker interface is a special interface with no methods or fields, used to convey metadata or a signal to the Java runtime or frameworks that a class has a specific capability or should be handled in a certain way. It acts like a "flag" or "tag" for the class.

Code :# marker interface

```
@FunctionalInterface
interface MyInterface {
    void doSomething();
}
```

What is difference between the abstract class and interface ?

An abstract class is a class that cannot be directly used to create objects.
It can have both completed methods (with code) and incomplete methods (without code).
It's used as a base for other classes to build on and share common behavior.

@MrJavaDecode

What is Encapsulation ?

Encapsulation is an object-oriented programming principle that binds data (fields) and the methods that operate on that data into a single unit — typically a class — while restricting direct access to some of the object's components.

It is achieved by declaring class variables as private and providing public getter and setter methods to access and modify those variables in a controlled manner.

Key Points:

- private variables can't be accessed directly.
- get and set methods allow controlled access.
- Helps in data protection and code maintainability.

@MrJavaDecode

#Encapsulation Code

```
public class Employee {
    private String name;    // hidden (private)
    private int salary;

    // Public methods to access or update (get/set)
    public String getName() {
        return name;
    }
    public void setName(String newName) {
        name = newName;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int newSalary) {
        if (newSalary > 0) {
            salary = newSalary;
        }
    }
}
```

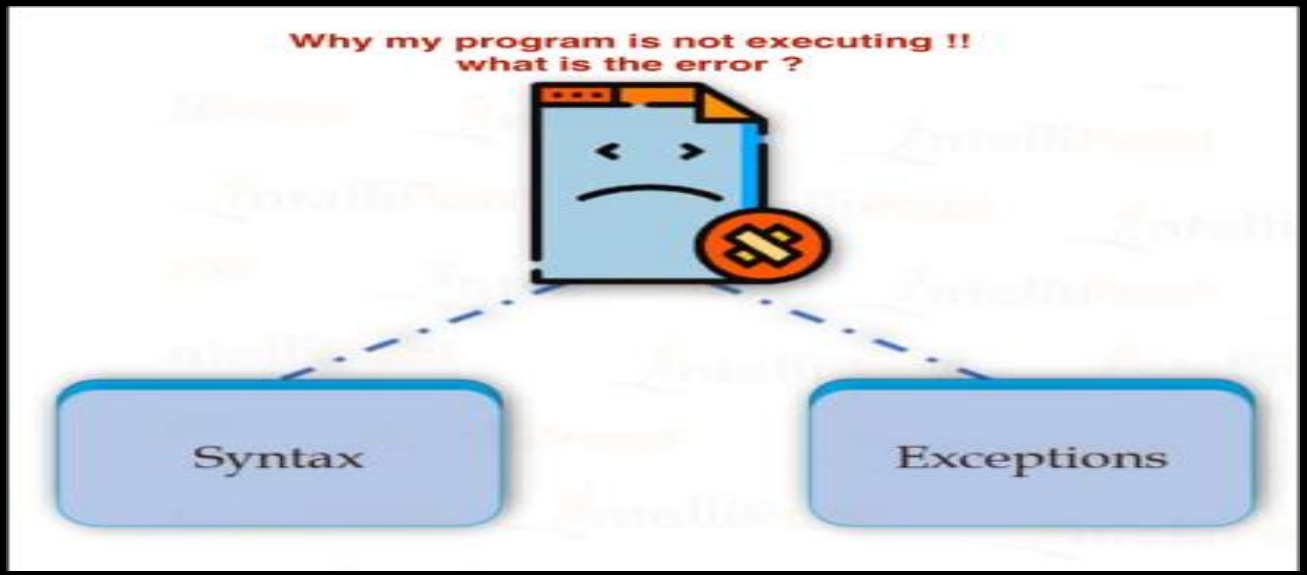
@MrJavaDecode

What is composition in java ?

composition is a design principle where one class contains a reference to another class — forming a *"has-a"* relationship. It's a way to build complex types by combining objects, rather than inheriting from a parent class.

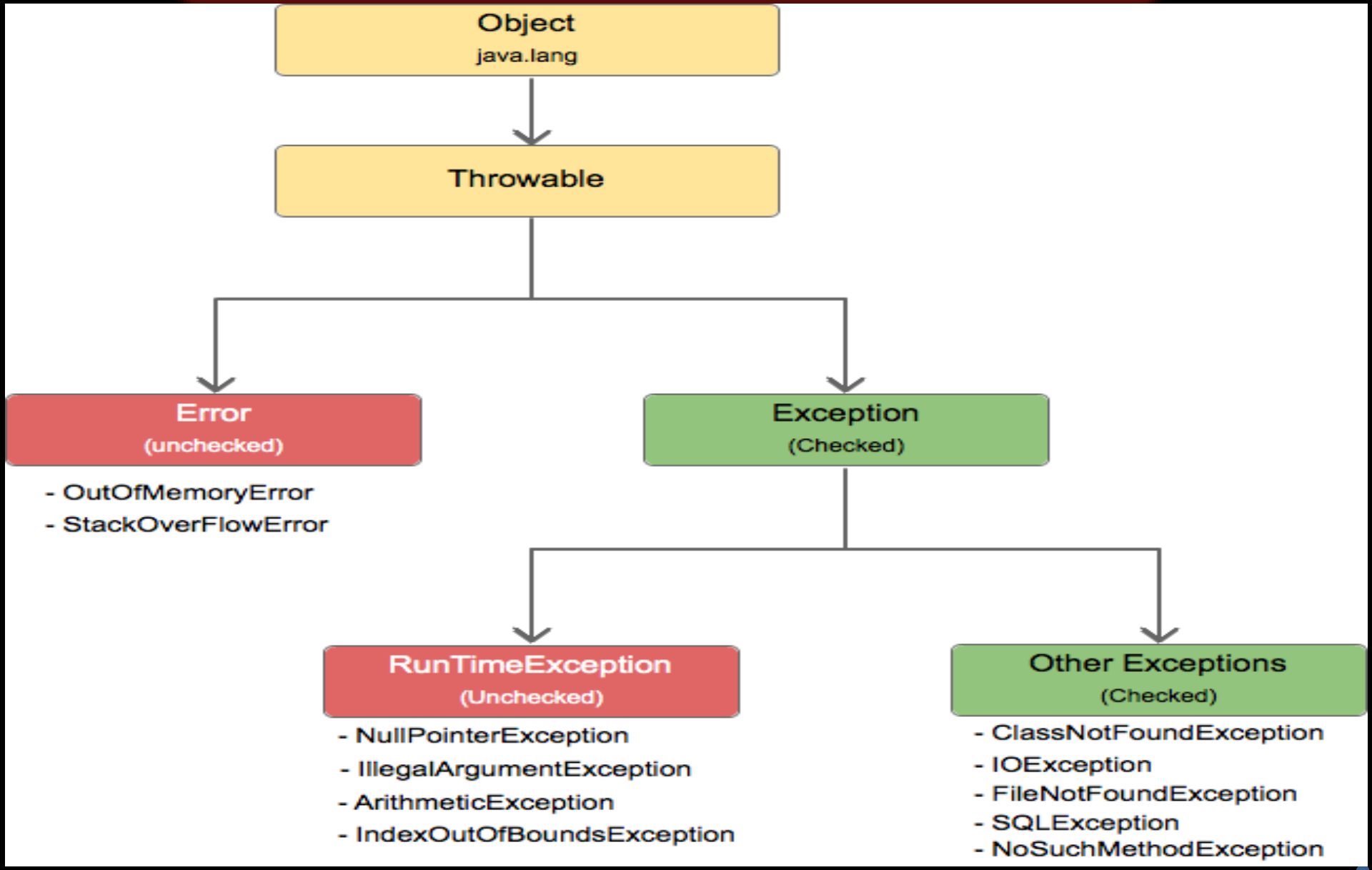
Explain exception handling in JAVA ?

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such a ClassNotFoundException, IOException, SQLException etc.



@MrJavaDecode

Exception hierarchy



@MrJavaDecode

java.lang.Object
↳ java.lang.Throwable
↳ java.lang.Error // Serious issues (usually not handled)
↳ java.lang.Exception // Can be handled in code
↳ CheckedException
↳ UncheckedException (Runtime Exception)

@MrJavaDecode

Exception Handling:

- **try:** Wraps code that might throw an exception.
- **catch:** Handles the exception.
- **finally:** Executes code regardless of whether an exception occurred.
- **throw:** Used to explicitly throw an exception.
- **throws:** Declares exceptions a method might throw.

```
1  package mrjavadecode.practice;
2
3  public class ExceptionExample {
4      public static void main(String[] args) {
5          try {
6              int result = 10 / 0; // This will throw ArithmeticException
7          } catch (ArithmeticException e) {
8              System.out.println("Cannot divide by zero!");
9          } finally {
0              System.out.println("This block always executes.");
1          }
2      }
3  }
4
```

#Checked Exceptions:

These are checked at compile-time. You must either handle them with a try-catch block or declare them using throws.

Examples:
IOException
SQLException
ClassNotFoundException

#Unchecked Exceptions (Runtime Exceptions)

These are not checked at compile-time. They occur due to programming bugs and can be avoided with proper coding.

Examples:
NullPointerException
ArrayIndexOutOfBoundsException
ArithmeticException
IllegalArgumentException

@MrJavaDecode

What is hashset class and how to store the value?

The **HashSet** class in Java is part of the **java.util** package and is used to create a collection that contains **no duplicate elements**. It implements the Set interface and is backed internally by a **HashMap**.

Key Features of HashSet:

- Stores **unique values** only.
- **No insertion order is maintained**.
- Allows **null** values.
- Uses **hashing** for storage and retrieval.
- Not synchronized (not thread-safe by default)

How Does HashSet Store Values?

When you add an element to a HashSet, it:

1. Calculates the **hashcode** of the element.
2. Maps it to a **bucket** in the underlying HashMap.
3. Stores it if no **equal object** already exists in that bucket.

@MrJavaDecode

```java
package mrjavadecode.practice;

import java.util.HashSet;

public class Example {

    public static void main(String[] args) {

        HashSet<String> fruitsName = new HashSet<>();

        fruitsName.add("Apple");
        fruitsName.add("Mango");
        fruitsName.add("Banana");
        fruitsName.add("Apple"); //Duplicate will be ignored

        System.out.println("fruitsList : " + fruitsName);
    }
}
```

Output: fruitsList : [Apple, Mango, Banana]

@MrJavaDecode

How can you make a class immutable in JAVA ?

An immutable class is one whose objects cannot be modified once created.
Java's String class is a classic example of immutability.

Steps to Create an Immutable Class in Java:

1. **Declare the class as final** :This prevents other classes from extending it and potentially altering its behavior.
2. **Make all fields private and final**
   This ensures fields are set only once, typically in the constructor.
3. **Do not provide setters**
   Setters allow modification, which breaks immutability.
4. **Initialize all fields via constructor**
   Assign values only once during object creation.
5. **Perform deep copies of mutable objects**
   If your class has fields like List, Map, or custom objects, clone them in the constructor and in getter methods to avoid exposing internal state.
6. **Return copies in getters for mutable fields**
   Never return the actual reference of a mutable field.

@MrJavaDecode

Code Example :

```java
package mrjavadecode.practice;

import java.util.ArrayList;
import java.util.List;

public final class Employee {

    private final String name;
    private final List<String> hobbies;

    public Employee(String name, List<String> hobbies) {
        this.name = name;
        this.hobbies = new ArrayList<>(hobbies); // deep copy
    }

    public String getName() {
        return name;
    }

    public List<String> getHobbies() {
        return new ArrayList<>(hobbies); // return copy
    }
}
/*
 * Now, even if someone tries to modify the list returned by getHobbies(), it
 * won't affect the original list inside the Person object.
 */

```

@MrJavaDecode

What is dispatcher servlet ?

In Spring Boot (and Spring MVC), the DispatcherServlet is the front controller responsible for handling all incoming HTTP requests and routing them to the appropriate components — such as controllers, views, and exception handlers.

How It Works:

When a request hits your Spring Boot app:

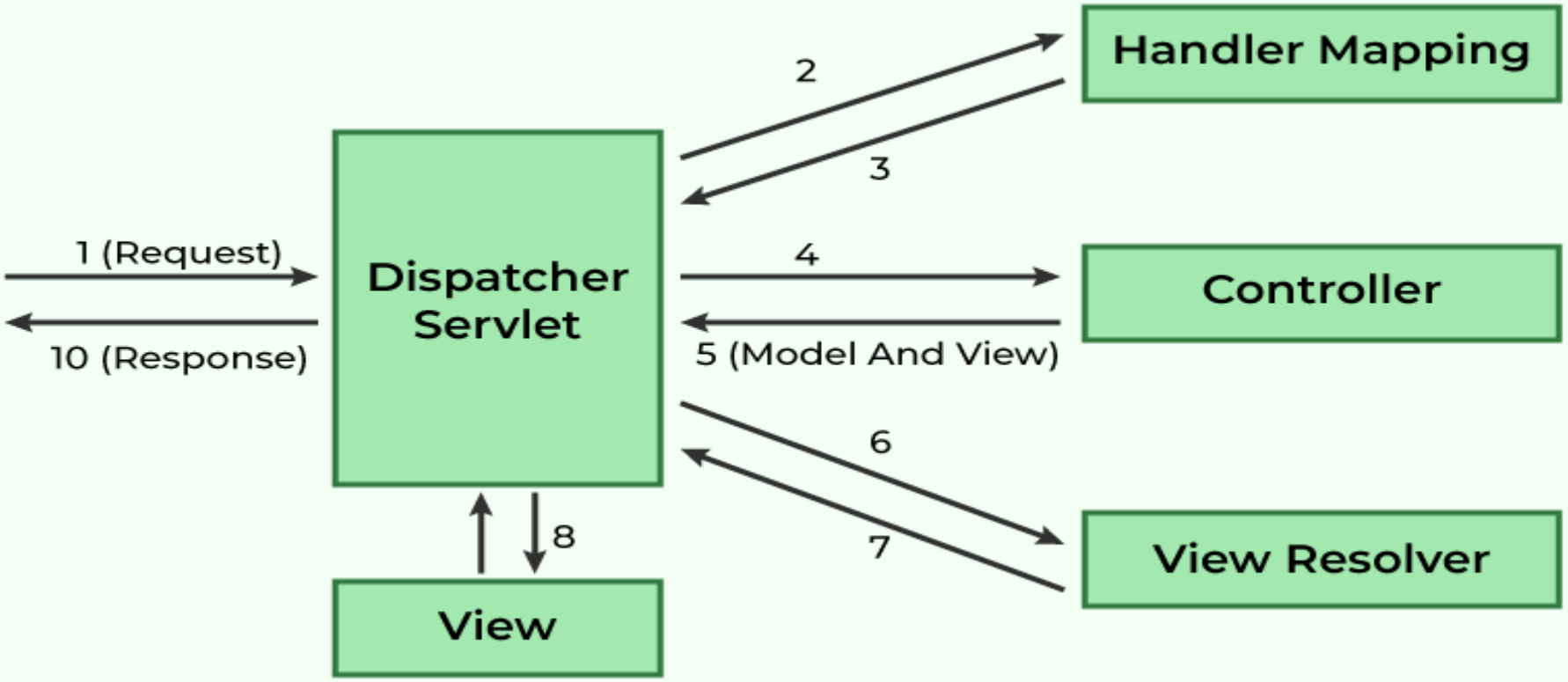
1. DispatcherServlet intercepts it.
2. It consults Handler Mapping to find the right controller method.
3. It invokes that method using a HandlerAdapter.
4. The controller returns a ModelAndView (or just data for REST).
5. DispatcherServlet uses a ViewResolver to render the response (for web pages) or returns JSON/XML (for REST APIs).

```java
@RestController
public class HelloController {
        @GetMapping("/hello")
                public String sayHello() {
        return "Hello, Shashank!";
  }
}
```

@MrJavaDecode

When you hit /hello, the DispatcherServlet:

- Receives the request
- Finds HelloController.sayHello()
- Executes it
- Returns "Hello, Shashank!" as the response



What is bean factory & Bean scope ?

A. BeanFactory is the core container of the Spring Framework responsible for managing and instantiating beans. It follows the IoC (Inversion of Control) principle — meaning Spring controls the lifecycle and dependencies of application objects.

it is part of the Spring Container and provides basic functionalities like:

- Creating and managing beans
- Handling dependency injection
- Providing lazy loading by default

B. Bean scope defines the lifecycle and visibility of a Spring bean — i.e., how many instances of a bean are created and where.

Common bean scope :

Singleton
Prototype
Request
Session
application

@MrJavaDecode

What is @RequestParam and @PathVariable ?

@RequestParam : is used to extract query parameters from the URL. It binds the value from the URL query string to a method parameter.

@PathVariable : is used to extract values from the URI path itself. It's ideal for RESTful APIs where resource identifiers are part of the URL.

Annotation	Extracts From	Example URL	Use Case
@RequestParam	Query Parameter	/greet?name=Shashank	Filtering, optional inputs
@PathVariable	URL Path Segment	/user/101	Resource identifiers in REST APIs

@MrJavaDecode

What is @RestController & @Controller ?

Feature	@Controller	@RestController
Purpose	Handles web requests and returns view (HTML/JSP)	Handles web requests and returns data (JSON/XML)
Returns	View name (resolved by ViewResolver)	Direct response body (data)
Annotation Type	Part of Spring MVC	@Controller + @ResponseBody
Common Use Case	MVC Web Applications (UI rendering)	REST APIs / Microservices
View Resolver Involved	Yes	No
HTTP Response Format	HTML or template-based page	JSON (default) or XML
Requires @ResponseBody?	✅ Yes (for returning data)	❌ No (included by default)
Example Return	"home" (renders home.html)	"Hello" (returns as raw response)

@MrJavaDecode

@MrJavaDecode

Code P1:

Find the second highest salary of an employee in MySQL.

```
SELECT DISTINCT salary
FROM employee
ORDER BY salary DESC
LIMIT 1, 1;
```

Code P2: Reverse the String by stream API ?

```
package mrjavadecode.practice;

import java.util.Arrays;
import java.util.Collections;
import java.util.stream.Collectors;

public class ReverseWords {

    public static void main(String[] args) {
        String sentence = "Hello MrJavaDecode !!";

        String reversed = Arrays.stream(sentence.split(" "))
                .collect(Collectors.collectingAndThen(Collectors.toList(), list -> {
                    Collections.reverse(list);
                    return String.join(" ", list);
                }));

        System.out.println("Reversed Sentence: " + reversed);
    }

}
```

Code P3 : Count Character Occurrence from Fruit Names:

```java
package mrjavadecode.practice;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

public class FruitLetterFrequency {

    public static void main(String[] args) {
        List<String> fruits = Arrays.asList("Apple", "Banana", "Orange");

        Map<Character, Long> letterFrequency = fruits.stream()
                .map(String::toLowerCase) // convert to lowercase
                .flatMapToInt(String::chars) // convert to IntStream of characters
                .mapToObj(c -> (char) c) // convert int to char
                .filter(Character::isLetter) // ignore spaces or symbols if any
                .collect(Collectors.groupingBy(Function.identity(), // group by each character
                        Collectors.counting() // count occurrences
                ));

        System.out.println("Letter Frequencies: " + letterFrequency);
    }

}
```

Console

<terminated> FruitLetterFrequency [Java Application] C:\Users\shash\Downloads\spring-tools-for-eclipse-4.31.0.RELEASE-e4.36.0-win32.win32.x86_64\sts-4.31.0.RELEASE

```
Letter Frequencies: {p=2, a=5, r=1, b=1, e=2, g=1, l=1, n=3, o=1}
```

@MrJavaDecode

Follow @MrJavaDecode

Thank You

@MrJavaDecode