Japneet Sachdeva • 2nd
Become a successful SDET with my Road To Full Stack QA Courses, Guides & Newsletters
Visit my store
1d •

Follow

Interview Questions from my last 3 Interview Experience January & Feburary 2025

1) Java program to remove duplicates characters from given String.

2) Program Remove the second highest element from the HashMap.

3) Java program to Generate prime numbers between 1 & given 4

number

4) How to find the missing values from a sorted array.

5) Java program to input name, middle name and surname of a

person and print only the initials.

5) Program to Print all Treemap elements?

6) What is a singleton Design Pattern? How do you implement that in your framework?

7) Write the Top 5 test cases for Booking Coupons.

8) What is serialization and deserialization?

9) What is the Difference between status codes 401 and 402? 1

10) Difference between selenium 3 and selenium 4?

11) What is delegate in Java and where do you use Delegate in your Framework?

12) How many maximum thread-pool can you open in the TestNG?

13) What are the Major challenges that come into the picture when you do parallel testing using TestNG and Grid?

14) How do you integrate your automation framework with the Jenkins pipeline?

15) What will happen if we remove the main method from the java program?

16) What is the component of your current Project?

17) How do you pass parameters in TestNG?

18) Write the logic of retrying the failed test case with a minimum 3 numbers of time in Automation Testing. Which Interface do you use for it?

19) What is the OOPs concept in java?

20) What is encapsulation?

21) What is Polymorphism?

22) Difference Between Classes and Objects?

23) What is collection in Java?

24) What is out in System.out.println?

25) In How many ways can we create an object?

26) Why is Java not 100% Object-oriented?

27) Can we make a constructor as Static?

28) How to convert a JSON object to a java object using Jackson?

29) What is the difference between Abstraction Class and Interfaces?

30) Difference between String, StringBuilder, and Stringbuffer?

31) What are other immutable classes in Java apart from String?

32) Difference between TreeMap and HashMap?

33) How do you set priorities for test automation, which test needs to be automated first?

34) How do you set test case priorities for your team?

35) What are the functional things you need to test on e-commerce sites?

36 Find Longest substring without repeating characters using both HashMap and HashSet Techniques

🌟 Note: Some questions might look simple, but the way you answer such questions makes the difference.

-x-x-

Solutions and how to answer such questions can be found here: **https://lnkd.in/gSU-m2F7**

#japneetsachdeva

Java OOP Concepts Cheat Sheet

Inheritance

- ✓ Inheritance, as name itself suggests, is used to inherit properties from parent class to child class.
- ✓ Using inheritance, you can reuse existing tried and tested code.
- ✓ Using inheritance, you can also add more features to existing class without modifying it by extending it through its subclass.
- ✓ In Java, inheritance is implemented by using extends keyword.
- ✓ An example :

```
class SuperClass
{
    String superClassField = "Super_Class_Field";

    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    String subClassField = "Sub_Class_Field";

    void subClassMethod()
    {
        System.out.println("Sub_Class_Method ");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SubClass subClass = new SubClass();

        subClass.subClassMethod();
        System.out.println(subClass.subClassField);

        //SuperClass properties are inherited to
SubClass

        subClass.superClassMethod();
        System.out.println(subClass.superClassField);
    }
}
```

Abstraction

- ✓ In computer science terms, abstraction means separating ideas from their actual implementations.
- ✓ Using abstraction, you define only ideas in one class so that those ideas can be implemented by its subclasses according to their requirements.
- ✓ In Java, abstraction is implemented by abstract classes and interfaces.
- ✓ An abstract Class example :

```
abstract class AbstractClass
{
    abstract void anidea();
}

class SubClassOne extends AbstractClass
{
    @Override
    void anidea()
    {
        System.out.println("An idea is implemented
according to SubClassOne requirement");
    }
}

class SubClassTwo extends AbstractClass
{
    @Override
    void anidea()
    {
        System.out.println("An idea is implemented
according to SubClassTwo requirement");
    }
}
```

✓ An interface example :

```
interface Interface
{
    void anidea();
}

class ClassOne implements Interface
{
    @Override
    public void anidea()
    {
        System.out.println("An idea is implemented
according to ClassOne requirement");
    }
}

class ClassTwo implements Interface
{
    @Override
    public void anidea()
    {
        System.out.println("An idea is implemented
according to ClassTwo requirement");
    }
}
```

Polymorphism

- ✓ Poly means many and morphs means forms. So, anything which has multiple forms is called as polymorphism.
- ✓ In computer science terms, any entity like operator or method or constructor which takes many forms and can be used for multiple tasks is called as polymorphism.
- ✓ For example, '+' operator can be used for addition of two numbers as well as for concatenation of two strings.
- ✓ In Java, there are two types of polymorphism - static polymorphism and dynamic polymorphism.
- ✓ Any entity which shows polymorphism during compilation is called static polymorphism.
- ✓ Operator overloading, method overloading and constructor overloading are best examples of static polymorphism.

```
class AnyClass
{
    int i;
    String s;

    //Constructor Overloading

    public AnyClass()
    {
        this.i = 1;
        this.s = "";
    }

    public AnyClass(int i, String s)
    {
        this.i = i;
        this.s = s;
    }

    //Method Overloading

    void anyMethod(int i)
    {
        System.out.println(i+this.i);   //Here, '+' is
used to add two numbers
    }

    void anyMethod(String s)
    {
        System.out.println(s+this.s);   //Here, '+' is
used to concatenate two strings
    }
}
```

- ✓ Any entity which shows polymorphism at run time is called as dynamic polymorphism.
- ✓ Method overriding is the best example of dynamic polymorphism.

```
class SuperClass
{
    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    @Override
    void superClassMethod()
    {
System.out.println("Super_Class_Method_Is_Overri
dden");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SuperClass superClass = new SuperClass();

        superClass.superClassMethod();
//Output : Super_Class_Method

        superClass = new SubClass();

        superClass.superClassMethod();
//Output : Super_Class_Method_Is_Overridden
    }
}
```

Encapsulation

- ✓ Bundling of data and operations to be performed on that data into single unit is called as encapsulation.
- ✓ Encapsulation in Java can be achieved by including both variables (data) and methods (operations) which act upon those variables into a single unit called class.
- ✓ Encapsulation is often used to hide important information from outside the world. It is called data hiding. This can be achieved by declaring all important variables as private and providing public getter and setter methods.

```
class Customer
{
    private int custID;
    private String name;
    private String address;

    //Getter and setter for custID

    public int getCustID()
    {
        return custID;
    }

    public void setCustID(int custID)
    {
        this.custID = custID;
    }

    //Getter and setter for name

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    //Getter and setter for address

    public String getAddress()
    {
        return address;
    }

    public void setAddress(String address)
    {
        this.address = address;
    }
}
```