Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Java Design Pattern Questions and Answers

Q. Exaplain MVC, Front-Controller, DAO, DTO, Service-Locator, Prototype design patterns?

2. Front-Controller

The front controller design pattern is used to provide a centralized request handling mechanism so that all requests will be handled by a single handler. This handler can do the authentication/ authorization/ logging or tracking of request and then pass the requests to corresponding handlers. Following are the entities of this type of design pattern.

- Front Controller - Single handler for all kinds of requests coming to the application (either web based/ desktop based).
- Dispatcher - Front Controller may use a dispatcher object which can dispatch the request to corresponding specific handler.
- View - Views are the object for which the requests are made.

Example:

We are going to create a `FrontController` and `Dispatcher` to act as Front Controller and Dispatcher correspondingly. `HomeView` and `StudentView` represent various views for which requests can come to front controller.

FrontControllerPatternDemo, our demo class, will use FrontController to demonstrate Front Controller Design Pattern.

Step 1 Create Views.

HomeView.java

```
public class HomeView {
   public void show(){
      System.out.println("Displaying Home Page");
   }
}
```

StudentView.java

```
public class StudentView {
   public void show(){
      System.out.println("Displaying Student Page");
   }
}
```

Step2 Create Dispatcher.

Dispatcher.java

```
public class Dispatcher {
   private StudentView studentView;
   private HomeView homeView;

   public Dispatcher(){
      studentView = new StudentView();
      homeView = new HomeView();
   }

   public void dispatch(String request){
      if(request.equalsIgnoreCase("STUDENT")){
         studentView.show();
      }
      else{
         homeView.show();
      }
   }
}
```

Step3 Create FrontController.

FrontController.java

```
public class FrontController {

   private Dispatcher dispatcher;

   public FrontController(){
      dispatcher = new Dispatcher();
   }

   private boolean isAuthenticUser(){
      System.out.println("User is authenticated successfully.");
      return true;
   }

   private void trackRequest(String request){
      System.out.println("Page requested: " + request);
   }

   public void dispatchRequest(String request){
      //log each request
      trackRequest(request);

      //authenticate the user
      if(isAuthenticUser()){
         dispatcher.dispatch(request);
      }
```

```
    }
}
```

Step4 Use the FrontController to demonstrate Front Controller Design Pattern.

FrontControllerPatternDemo.java

```
public class FrontControllerPatternDemo {
   public static void main(String[] args) {

      FrontController frontController = new FrontController();
      frontController.dispatchRequest("HOME");
      frontController.dispatchRequest("STUDENT");
   }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What are the design patterns available in Java?

Java Design Patterns are divided into three categories – creational, structural, and behavioral design patterns.

1. Creational Design Patterns

- Singleton Pattern
- Factory Pattern
- Abstract Factory Pattern
- Builder Pattern
- Prototype Pattern

2. Structural Design Patterns

- Adapter Pattern
- Composite Pattern
- Proxy Pattern
- Flyweight Pattern
- Facade Pattern
- Bridge Pattern
- Decorator Pattern

3. Behavioral Design Patterns

- Template Method Pattern
- Mediator Pattern
- Chain of Responsibility Pattern
- Observer Pattern
- Strategy Pattern
- Command Pattern

- State Pattern
- Visitor Pattern
- Interpreter Pattern
- Iterator Pattern
- Memento Pattern

4. Miscellaneous Design Patterns

- DAO Design Pattern
- Dependency Injection Pattern
- MVC Pattern


```
<b><a href="#">↥ back to top</a></b>
```


Q. Explain Singleton Design Pattern in Java?

1. Eager initialization:
In eager initialization, the instance of Singleton Class is created at the time of class loading.

Example:

```
public class EagerInitializedSingleton {

    private static final EagerInitializedSingleton instance = new
EagerInitializedSingleton();

    // private constructor to avoid client applications to use
constructor
    private EagerInitializedSingleton(){}

    public static EagerInitializedSingleton getInstance(){
        return instance;
    }
}
```

2. Static block initialization
Static block initialization implementation is similar to eager initialization, except that instance of class is created in the static block that provides option for exception handling.

Example:

```
public class StaticBlockSingleton  {

    private static StaticBlockSingleton  instance;

    private StaticBlockSingleton (){}

    // static block initialization for exception handling
    static{
        try{
            instance = new StaticBlockSingleton ();
```

```
        }catch(Exception e){
            throw new RuntimeException("Exception occured in creating
Singleton instance");
        }
    }

    public static StaticBlockSingleton getInstance(){
        return instance;
    }
}
```

3. Lazy Initialization

Lazy initialization method to implement Singleton pattern creates the instance in the global access method.

Example:

```
public class LazyInitializedSingleton  {

    private static LazyInitializedSingleton  instance;

    private LazyInitializedSingleton(){}

    public static LazyInitializedSingleton  getInstance(){
        if(instance == null){
            instance = new LazyInitializedSingleton ();
        }
        return instance;
    }
}
```

4. Thread Safe Singleton

The easier way to create a thread-safe singleton class is to make the global access method synchronized, so that only one thread can execute this method at a time.

Example:

```
public class ThreadSafeSingleton {

    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton(){}

    public static synchronized ThreadSafeSingleton getInstance(){
        if(instance == null){
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }
}
```

5. Bill Pugh Singleton Implementation
Prior to Java5, memory model had a lot of issues and above methods caused failure in certain scenarios in multithreaded environment. So, Bill Pugh suggested a concept of inner static classes to use for singleton.

Example:

```
public class BillPughSingleton {

    private BillPughSingleton(){}

    private static class SingletonHelper{
        private static final BillPughSingleton INSTANCE = new
BillPughSingleton();
    }

    public static BillPughSingleton getInstance(){
        return SingletonHelper.INSTANCE;
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. Explain Adapter Design Pattern in Java?

Adapter design pattern is one of the structural design pattern and its used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an Adapter.

Example:

we have two incompatible interfaces: **MediaPlayer** and **MediaPackage**. MP3 class is an implementation of the MediaPlayer interface and we have VLC and MP4 as implementations of the MediaPackage interface. We want to use MediaPackage implementations as MediaPlayer instances. So, we need to create an adapter to help to work with two incompatible classes.

MediaPlayer.java

```
public interface MediaPlayer {
    void play(String filename);
}
```

MediaPackage.java

```
public interface MediaPackage {
    void playFile(String filename);
}
```

MP3.java

```
public class MP3 implements MediaPlayer {
 @Override
 public void play(String filename) {
    System.out.println("Playing MP3 File " + filename);
 }
}
```

MP4.java

```
public class MP4 implements MediaPackage {
    @Override
    public void playFile(String filename) {
        System.out.println("Playing MP4 File " + filename);
    }
}
```

VLC.java

```
public class VLC implements MediaPackage {
    @Override
    public void playFile(String filename) {
        System.out.println("Playing VLC File " + filename);
    }
}
```

FormatAdapter.java

```
public class FormatAdapter implements MediaPlayer {
    private MediaPackage media;
    public FormatAdapter(MediaPackage m) {
        media = m;
    }
    @Override
    public void play(String filename) {
        System.out.print("Using Adapter --> ");
        media.playFile(filename);
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        MediaPlayer player = new MP3();
        player.play("file.mp3");
        player = new FormatAdapter(new MP4());
        player.play("file.mp4");
        player = new FormatAdapter(new VLC());
        player.play("file.avi");
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. Explain Factory Design Pattern in Java?

A Factory Pattern or Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance of the class.

Example: Calculate Electricity Bill Plan.java

```
import java.io.*;
abstract class Plan {
    protected double rate;
    abstract void getRate();

    public void calculateBill(int units){
        System.out.println(units*rate);
    }
}
```

DomesticPlan.java

```
class  DomesticPlan extends Plan{
    @override
    public void getRate(){
        rate=3.50;
    }
}
```

CommercialPlan.java

```
class  CommercialPlan extends Plan{
    @override
    public void getRate(){
        rate=7.50;
    }
}
```

InstitutionalPlan.java

```
class  InstitutionalPlan extends Plan{
    @override
    public void getRate(){
        rate=5.50;
   }
}
```

GetPlanFactory.java

```
class GetPlanFactory {

    // use getPlan method to get object of type Plan
```

```
    public Plan getPlan(String planType){
        if(planType == null){
            return null;
        }
        if(planType.equalsIgnoreCase("DOMESTICPLAN")) {
                return new DomesticPlan();
            }
        else if(planType.equalsIgnoreCase("COMMERCIALPLAN")){
            return new CommercialPlan();
        }
        else if(planType.equalsIgnoreCase("INSTITUTIONALPLAN")) {
            return new InstitutionalPlan();
        }
         return null;
    }
}
```

GenerateBill.java

```
import java.io.*;
class GenerateBill {

    public static void main(String args[])throws IOException {
      GetPlanFactory planFactory = new GetPlanFactory();

      System.out.print("Enter the name of plan for which the bill will
be generated: ");
      BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

      String planName=br.readLine();
      System.out.print("Enter the number of units for bill will be
calculated: ");
      int units=Integer.parseInt(br.readLine());

      Plan p = planFactory.getPlan(planName);
      // call getRate() method and calculateBill()method of
DomesticPaln.

       System.out.print("Bill amount for "+planName+" of  "+units+"
units is: ");
            p.getRate();
            p.calculateBill(units);
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. Explain Strategy Design Pattern in Java?

Strategy design pattern is one of the behavioral design pattern. Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime.

Example: Simple Shopping Cart where we have two payment strategies – using Credit Card or using PayPal.

PaymentStrategy.java

```
public interface PaymentStrategy {
    public void pay(int amount);
}
```

CreditCardStrategy.java

```
public class CreditCardStrategy implements PaymentStrategy {

    private String name;
    private String cardNumber;
    private String cvv;
    private String dateOfExpiry;

    public CreditCardStrategy(String nm, String ccNum, String cvv,
String expiryDate){
        this.name=nm;
        this.cardNumber=ccNum;
        this.cvv=cvv;
        this.dateOfExpiry=expiryDate;
    }
    @Override
    public void pay(int amount) {
        System.out.println(amount +" paid with credit/debit card");
    }
}
```

PaypalStrategy.java

```
public class PaypalStrategy implements PaymentStrategy {

    private String emailId;
    private String password;

    public PaypalStrategy(String email, String pwd){
        this.emailId=email;
        this.password=pwd;
    }
    @Override
    public void pay(int amount) {
        System.out.println(amount + " paid using Paypal.");
```

```
    }
}
```

Item.java

```
public class Item {

    private String upcCode;
    private int price;

    public Item(String upc, int cost){
        this.upcCode=upc;
        this.price=cost;
    }
    public String getUpcCode() {
        return upcCode;
    }
    public int getPrice() {
        return price;
    }
}
```

ShoppingCart.java

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {

    List<Item> items;

    public ShoppingCart(){
        this.items=new ArrayList<Item>();
    }
    public void addItem(Item item){
        this.items.add(item);
    }
    public void removeItem(Item item){
        this.items.remove(item);
    }
    public int calculateTotal(){
        int sum = 0;
        for(Item item : items){
            sum += item.getPrice();
        }
        return sum;
    }
    public void pay(PaymentStrategy paymentMethod){
        int amount = calculateTotal();
        paymentMethod.pay(amount);
```

```
    }
}
```

ShoppingCartTest.java

```
public class ShoppingCartTest {

    public static void main(String[] args) {
        ShoppingCart cart = new ShoppingCart();

        Item item1 = new Item("1234",10);
        Item item2 = new Item("5678",40);

        cart.addItem(item1);
        cart.addItem(item2);

        // pay by paypal
        cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));

        // pay by credit card
        cart.pay(new CreditCardStrategy("Pankaj Kumar",
"1234567890123456", "786", "12/15"));
    }
}
```

Output

```
500 paid using Paypal.
500 paid with credit/debit card
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. When do you use Flyweight pattern?

Q. What is difference between dependency injection and factory design pattern?

Q. Difference between Adapter and Decorator pattern?

Q. Difference between Adapter and Proxy Pattern?

Q. What is Template method pattern?

Q. When do you use Visitor design pattern?

Q. When do you use Composite design pattern?

Q. Difference between Abstract factory and Prototype design pattern?

```
<b><a href="#">↥ back to top</a></b>
```

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project