**Scenario 01**

**CODE**

import java.util.ArrayList;

public class Main{

    public static void main(String[] args) {

    System.out.println("\n\t23k-3032\n\tShah Hunain");

    ArrayList<String> skills1 = new ArrayList<>();

    skills1.add("Fishing");

    skills1.add("Acting");

    Pet pet1 = new Pet("Healthy",6,9,skills1);

    ArrayList<String> skills2 = new ArrayList<>();

    skills2.add("Responding");

    skills2.add("Skating");

    Pet pet2 = new Pet("Sick",5,8,skills2);

    Adopter adopter1 = new Adopter("Ali","01234567890");

    Adopter adopter2 = new Adopter("Ahmed","01231231230");

    adopter1.adoptPet(pet1);

    adopter2.adoptPet(pet2);

    adopter1.displayAdoptedPets();

    adopter2.displayAdoptedPets();

    pet1.updateHappiness(-3);

    pet1.updateHealth("Sick");

    pet1.updateHunger(2);

    pet1.displayPetDetails();

    pet2.updateHappiness(2);

    pet2.updateHealth("Healthy");

    pet2.updateHunger(-3);

    pet2.displayPetDetails();

    adopter1.returnPet(pet1);

    adopter2.returnPet(pet2);

    }

}

import java.util.ArrayList;

public class Adopter {

    private String adopterName;

    private String adopterMobileNum;

    private ArrayList<Pet> adoptedPetRecords;

    public Adopter(String adopterName, String adopterMobileNum) {

        this.adopterName = adopterName;

        this.adopterMobileNum = adopterMobileNum;

        this.adoptedPetRecords = new ArrayList<>();

    }

    public void adoptPet(Pet pet) {

        adoptedPetRecords.add(pet);

        System.out.println("Pet adopted successfully!");

    }

    public void returnPet(Pet pet) {

        adoptedPetRecords.remove(pet);

        System.out.println("Pet returned successfully!");

    }

    public void displayAdoptedPets() {

        System.out.println("Adopted Pets:");

        for (Pet pet : adoptedPetRecords) {

            pet.displayPetDetails();

        }

    }

}

import java.util.ArrayList;

public class Pet {

    private String healthStatus;

    private int hungerLevel;

    private int happinessLevel;

    private ArrayList<String> specialSkills;

    Pet(String healthStatus, int hungerLevel, int happinessLevel, ArrayList<String> specialSkills){

        this.healthStatus = healthStatus;

        this.hungerLevel = hungerLevel;

        this.happinessLevel = happinessLevel;

        this.specialSkills = specialSkills;

    }

    public void displayPetDetails(){

        System.out.println("Health Status: "+healthStatus);

        System.out.println("Hunger Level: "+hungerLevel);

        System.out.println("Happiness Level: "+happinessLevel);

        System.out.println("Special Skills: "+specialSkills);

    }

    public void updateHappiness(int amount){

        happinessLevel =happinessLevel + amount;

        if (happinessLevel>10){

            happinessLevel=10;

        }

        else if(happinessLevel<0){

            happinessLevel=0;

        }

    }

    public void updateHealth(String newHealthStatus){

        healthStatus = newHealthStatus;

    }

    public void updateHunger(int amount){

        hungerLevel =hungerLevel + amount;

        if (hungerLevel > 10){

            hungerLevel = 10;

        }

        else if(hungerLevel<0){

            hungerLevel = 0;

        }

        else{

        }

        if(amount<0){

            updateHappiness(1);

        }

        else if(amount>0){

            updateHappiness(-1);

        }

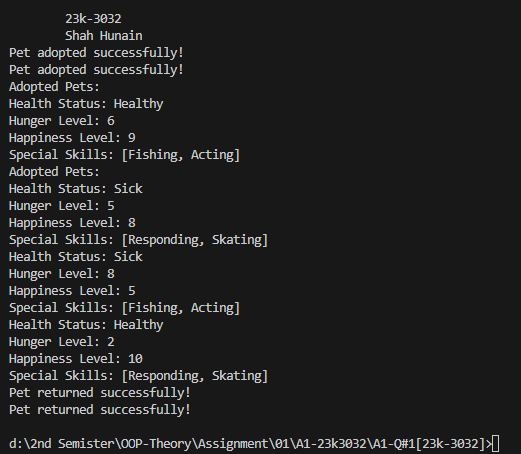
        else{

        }

    }

}

**OUTPUT**



**Scenario 02**

**CODE**

public class Main {

    public static void main(String[] args) {

System.out.println("\n\t23k-3032\n\tShah Hunain");

    Table[] tables = new Table[5];

    tables[0] = new Table(8);

    tables[1] = new Table(8);

    tables[2] = new Table();

    tables[3] = new Table();

    tables[4] = new Table();

    occupyTable(tables, 4);

    occupyTable(tables, 6);

    tables[0].haveLunch();

    System.out.println("Table 1 status:");

    System.out.println("Clean? " + tables[0].isClean());

    tables[0].haveLunch();

    System.out.println("Table 1 status:");

    System.out.println("Clean= " + tables[0].isClean());

    tables[0].leave();

    System.out.println("Table 1 status:");

    System.out.println("Occupied Seats= " + tables[0].getOccupiedSeats());

    tables[0].cleanTable();

    System.out.println("Table 1 status:");

    System.out.println("Clean? " + tables[0].isClean());

    emptyTable(tables, 1);

    }

    public static void occupyTable(Table[] tables, int groupSize) {

        for(Table table : tables){

            if(table.occupy(groupSize)) {

                System.out.println(""+groupSize+" are seated on the table whose capacity is: "+table.getCapacity());

                return;

            }

        }

        System.out.println("Table not available for "+groupSize);

    }

    public static void emptyTable(Table[] tables, int tableIndex){

        if (tableIndex >= 0 && tableIndex <tables.length){

            tables[tableIndex].leave();

            tables[tableIndex].cleanTable();

            System.out.println("Table "+tableIndex+" emptied");

        }

        else{

            System.out.println("Invalid Index.");

        }

    }

}

public class Table{

    private final int capacity;

    private int occupiedSeats;

    private boolean clean;

    Table() {

    this.capacity=4;

    this.occupiedSeats=0;

    this.clean=true;

    }

    Table(int capacity){

        if(capacity==4 || capacity==8) {

            this.capacity=capacity;

        }

        else if(capacity<6){

            this.capacity=4;

        }

        else{

            this.capacity=8;

        }

        this.occupiedSeats=0;

        this.clean=true;

    }

    public int getCapacity(){

        return capacity;

    }

    public int getOccupiedSeats(){

        return occupiedSeats;

    }

    public int getFreeSeats(){

        return (capacity-occupiedSeats);

    }

    public boolean isClean(){

        return clean;

    }

    public boolean occupy(int size){

        if (clean && size<=capacity){

            occupiedSeats=size;

            clean=false;

            return true;

        }

        return false;

    }

    public void haveLunch() {

        clean=false;

    }

    public void leave() {

        occupiedSeats=0;

    }

    public void cleanTable() {

        if (occupiedSeats==0){

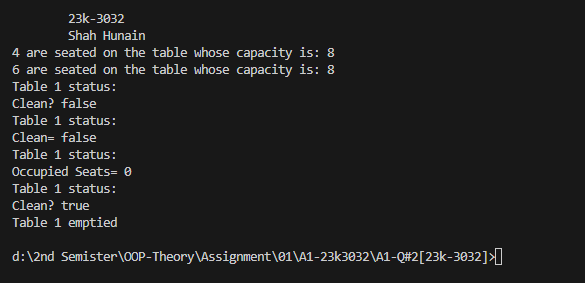
            clean=true;

        }

    }

}

**OUTPUT**



**Scenario 03**

**CODE**

public class Main {

    public static void main(String[] args) {

    System.out.println("\n\t23k-3032\n\tShah Hunain");

    ChessBoard chessBoard = new ChessBoard();

    System.out.println("Initial state:");

    chessBoard.display();

    boolean isValidMove1 = chessBoard.movePiece("b8", "a6");

    System.out.println("Valid?   "+isValidMove1);

    System.out.println("After moving knight from b8 to a6:\n");

    chessBoard.display();

    boolean isValidMove2 = chessBoard.movePiece("g7", "g5");

    System.out.println("Valid?   "+isValidMove2);

    System.out.println("State after moving pawn from g7 to g5:\n");

    chessBoard.display();

    }

}

public class ChessPiece {

    private String name;

    private String color;

    private char symbol;

    ChessPiece(){

        this.name="Pawn";

        this.color="White";

        this.symbol='p';

    }

    ChessPiece(String name, String color, char symbol){

        this.name=name;

        this.color=color;

        this.symbol=symbol;

    }

    public String getName(){

        return name;

    }

    public void setName(String name){

        this.name=name;

    }

    public String getColor(){

        return color;

    }

    public void setColor(String color){

        this.color=color;

    }

    public char getSymbol(){

        return symbol;

    }

    public void setSymbol(char symbol){

        this.symbol=symbol;

    }

}

public class ChessBoard {

    private ChessPiece[][] board;

    ChessBoard(){

        initializeBoard();

    }

    private void initializeBoard(){

        board = new ChessPiece[8][8];

        for (int j=0; j<8; j++){

            board[1][j] = new ChessPiece("Pawn", "Black", 'P');

            board[6][j] = new ChessPiece("Pawn", "White", 'p');

        }

    }

    public void display(){

        System.out.println("  a b c d e f g h");

        for (int j=0; j<8; j++){

            System.out.print((8-j) + " ");

            for (int k = 0; k<8; k++){

                if (board[j][k]==null) {

                    System.out.print(". ");

                }

                else{

                    System.out.print(" "+board[j][k].getSymbol());

                }

            }

            System.out.println(8-j);

        }

        System.out.println("  a b c d e f g h");

    }

    public boolean movePiece(String source, String destination){

        int sourceRow= 8 - Character.getNumericValue(source.charAt(1));

        int sourceCol= source.charAt(0)-'a';

        int destRow= 8 - Character.getNumericValue(destination.charAt(1));

        int destCol= destination.charAt(0)-'a';

    if (sourceRow < 0 || sourceRow > 7 || sourceCol < 0 || sourceCol > 7 || destRow < 0 || destRow > 7 || destCol < 0 || destCol > 7){

            System.out.println("Invalid Move");

            return false;

        }

        if (board[sourceRow][sourceCol]==null) {

            System.out.println("No piece at Source");

            return false;

        }

        if (board[destRow][destCol]!= null) {

            System.out.println("Destination position is already Occupied.");

            return false;

        }

        int rowDiff = Math.abs(destRow - sourceRow);

        int colDiff = Math.abs(destCol - sourceCol);

        ChessPiece piece = board[sourceRow][sourceCol];

        if ((rowDiff==2 && colDiff==1) || (rowDiff==1 && colDiff==2)){

            board[destRow][destCol]=piece;

            board[sourceRow][sourceCol]=null;

            return true;

        }

        else if(piece.getName().equals("Pawn")){

            if((piece.getColor().equals("White") && sourceRow==6 && rowDiff<=2 && colDiff==0) || (piece.getColor().equals("Black") && sourceRow==1 && rowDiff<=2 && colDiff==0)){

                board[destRow][destCol]=piece;

                board[sourceRow][sourceCol]=null;

                return true;

            }

            else{

                System.out.println("Invalid Move");

                return false;

            }

        }

        else {

            System.out.println("Invalid Move");

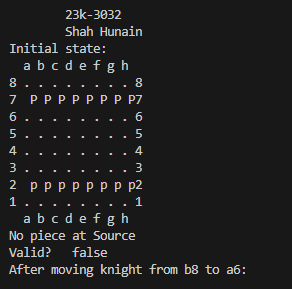
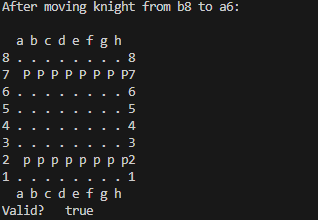
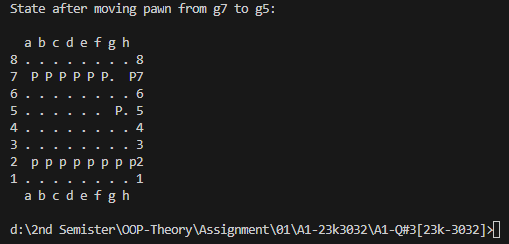
            return false;

        }

    }

}

**OUTPUT**

**Scenario 04**

**CODE**

import java.util.Scanner;

public class Main{

    public static void main(String[] args) {

        System.out.println("\n23k3032\nShah Hunain\n");

        Scanner obj = new Scanner(System.in);

        System.out.println("Enter roller coaster details:");

        System.out.print("Name (of the attraction- some creative name): ");

        String name = obj.nextLine();

        System.out.print("Height (maximum height that the roller coaster can reach): ");

        int height = obj.nextInt();

        obj.nextLine();

        System.out.print("Length (total length of the roller coaster track): ");

        int length = obj.nextInt();

        obj.nextLine();

        System.out.print("Speed: ");

        int speed = obj.nextInt();

        obj.nextLine();

        System.out.print("Capacity (amount of people that can be seated at once): ");

        int capacity = obj.nextInt();

        obj.nextLine();

        System.out.print("Current number of riders (number of passengers/riders currently seated in the roller coaster): ");

        int riders = obj.nextInt();

        obj.nextLine();

        System.out.print("Is ride in progress? (1=Yes\n0=No): ");

        boolean inProgress = obj.nextInt() == 1;

        Park theme = new Park(name, height, length, speed, capacity, riders, inProgress);

        theme.load();

        theme.start();

        theme.stop();

        theme.unload();

        theme.accelerate();

        theme.brakes();

    }

}

public class Park {

    private String name;

    private int height;

    private int length;

    private int speed;

    private int capacity;

    private int currentNumRiders;

    private boolean rideInProgress;

    Park(){

        this.name="Roller Coaster";

        this.height=500;

        this.length=2000;

        this.speed=0;

        this.capacity=20;

        this.rideInProgress=false;

        this.adjustCapacity();

    }

    Park(String n, int h, int l, int s, int c, int riders, boolean inProgress){

        this.name=n;

        this.height=h;

        this.length=l;

        this.speed=s;

        this.capacity=c;

        this.currentNumRiders=riders;

        this.rideInProgress=inProgress;

        this.adjustCapacity();

    }

    private void adjustCapacity(){

        if (capacity>3){

            if(capacity %2 != 0 || capacity % 3 != 0){

                capacity = capacity + 1;

            }

        }

        else{

            System.out.println("Invalid capacity");

        }

    }

    public void setName(String n){

        name=n;

    }

    public void setHeight(int h){

        height=h;

    }

    public void setLength(int l){

        length=l;

    }

    public void setSpeed(int s){

        speed=s;

    }

    public void setCapacity(int c){

        capacity = c;

        adjustCapacity();

    }

    public void setCurrentNumRiders(int r){

        currentNumRiders=r;

    }

    public void setRideInProgress(boolean p){

        rideInProgress=p;

        adjustCapacity();

    }

    public String getName(){

        return name;

    }

    public int getHeight(){

        return height;

    }

    public int getLength(){

        return length;

    }

    public int getSpeed(){

        return speed;

    }

    public int getCapacity(){

        return capacity;

    }

    public int getCurrentNumRiders(){

        return currentNumRiders;

    }

    public boolean isRideInProgress(){

        return rideInProgress;

    }

    public void load(){

        if (!rideInProgress){

            if(currentNumRiders<=capacity){

                System.out.println("Passengers successfully seated");

            }

            else{

                int rem = currentNumRiders - capacity;

                System.out.println("Number of passengers not seated successfully: " + rem);

            }

        }

        else{

            System.out.println("Ride is in progress");

        }

    }

    public void start() {

        if (!rideInProgress) {

            int remainingSeats = capacity - currentNumRiders;

            System.out.println("Seats Remaining: " + remainingSeats);

        }

    }

    public void stop() {

        if (rideInProgress) {

            rideInProgress = false;

            System.out.println("Ride stopped");

        } else {

            System.out.println("Ride is already stopped");

        }

    }

    public void unload() {

        if (!rideInProgress) {

            System.out.println("Passengers unloaded");

        } else {

            System.out.println("Ride is in progress, passengers cannot be unloaded");

        }

    }

    public void accelerate() {

        int lastDigit = speed % 10;

        speed = speed + lastDigit;

        System.out.println("Speed increased to "+speed);

    }

    public void brakes() {

        if (speed != 0) {

            int tempSpeed = speed;

            while (tempSpeed >= 10){

                tempSpeed /= 10;

            }

            speed =speed - tempSpeed;

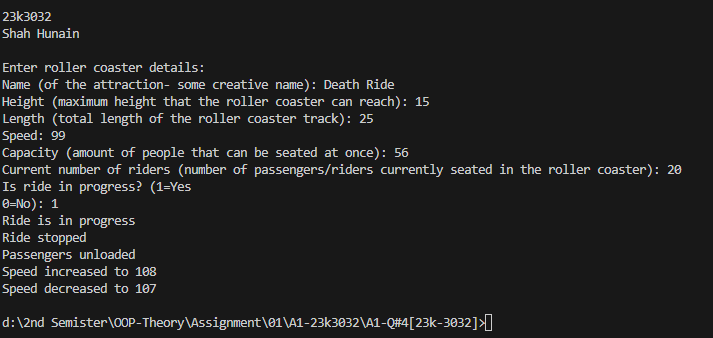
            System.out.println("Speed decreased to "+speed);

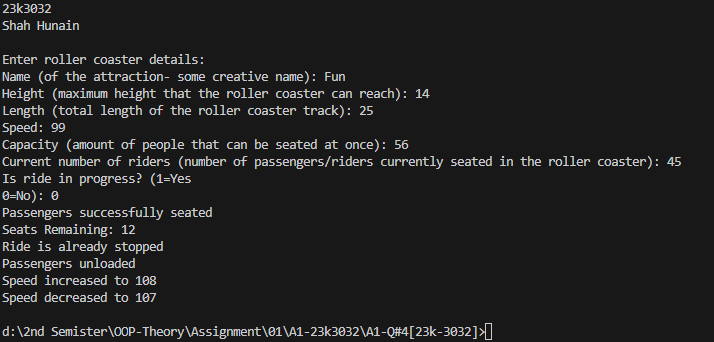
        }

    }

}

**OUTPUT**





**Scenario 05**

**CODE**

import java.util.ArrayList;

public class Main{

    public static void main(String[] args){

        System.out.println("\n23k3032\nShah Hunain\n");

        BOGOCoupon fHc = new BOGOCoupon("FH-BOGO-12345", 5, 100, "FH");

        BOGOCoupon pBc = new BOGOCoupon("PB-BOGO-67890", 3, 150, "PB");

        ArrayList<String> fHm = new ArrayList<>();

        fHm.add("Sushi");

        fHm.add("Pad Thai");

        fHm.add("Mango Tango");

        ArrayList<Integer> fHp = new ArrayList<>();

        fHp.add(2500);

        fHp.add(2000);

        fHp.add(1000);

        ArrayList<String> foodHavenCoupons = new ArrayList<>();

        foodHavenCoupons.add("FH-BOGO-12345");

        Restaurant foodHaven = new Restaurant("Food Haven", fHm, fHp, foodHavenCoupons);

        User user = new User("Ali", 17, "123456789");

        user.accumulateCoupon("FH-BOGO-12345");

        foodHaven.displayMenu();

        int currentDate = 20; // 20th day of the year

        if (user.hasCoupon("FH-BOGO-12345") && fHc.isValid(currentDate, "FH")) {

            System.out.println("Valid coupon for Food Haven!");

        } else {

            System.out.println("Invalid coupon for Food Haven.");

        }

    }

}

public class BOGOCoupon{

    private String couponCode;

    private int validFrom;

    private int validUntil;

    private String restaurantCode;

    public BOGOCoupon(String couponCode, int validFrom, int validUntil, String restaurantCode) {

        this.couponCode = couponCode;

        this.validFrom = validFrom;

        this.validUntil = validUntil;

        this.restaurantCode = restaurantCode;

    }

    public boolean isValid(int currentDate, String restaurantCode) {

        return ((currentDate >= validFrom) && (currentDate <= validUntil) && this.restaurantCode.equals(restaurantCode));

    }

}

import java.util.ArrayList;

public class Restaurant{

    private String restaurantName;

    private ArrayList<String> menuList;

    private ArrayList<Integer> priceList;

    private ArrayList<String> validCouponCodesList;

    Restaurant(String n, ArrayList<String> menuList, ArrayList<Integer> priceList, ArrayList<String> validCouponCodesList) {

        this.restaurantName = n;

        this.menuList = menuList;

        this.priceList = priceList;

        this.validCouponCodesList = validCouponCodesList;

    }

    public void displayMenu() {

        System.out.println("\t" + restaurantName + " welcomes you!");

        System.out.println("\n\tThe MENU is:");

        int n = menuList.size();

        for (int j = 0; j < n; j++) {

            System.out.println(menuList.get(j) + ": Rs" + priceList.get(j));

        }

    }

    public boolean isValidCoupon(String code) {

        return (validCouponCodesList.contains(code));

    }

}

import java.util.ArrayList;

public class User{

    private String name;

    private int age;

    private String mobileNumber;

    private ArrayList<String> couponsList;

    User() {

        this.name = "null";

        this.age = 0;

        this.mobileNumber = "null";

    }

    User(String name, int age, String mobileNumber) {

        this.name = name;

        this.age = age;

        this.mobileNumber = mobileNumber;

        this.couponsList = new ArrayList<>();

    }

    public void accumulateCoupon(String coupon) {

        couponsList.add(coupon);

    }

    public boolean hasCoupon(String coupon) {

        return (couponsList.contains(coupon));

    }

}

**OUTPUT**

