1. Define an interface “IntOperations” with methods to check whether an  
integer is positive\_negative, even\_odd, prime and operations like  
factorial and sum of digits. Define a class MyNumber having one  
private int data member. Write a default constructor to initialize it to 0  
and another constructor to initialize it to a value (Use this). Implement  
the above interface. Create an object in main.  
interface IntOperations {  
boolean isPositive();  
boolean isNegative();  
boolean isEven();  
boolean isOdd();  
boolean isPrime();  
long factorial();  
int sumOfDigits();  
}  
class MyNumber implements IntOperations {  
private int data;  
public MyNumber() {  
this.data = 0;  
}  
public MyNumber(int data) {  
this.data = data;  
}  
@Override  
public boolean isPositive() {  
return data > 0;  
}  
@Override  
public boolean isNegative() {  
return data < 0;  
}

@Override  
public boolean isEven() {  
return data % 2 == 0;  
}  
@Override  
public boolean isOdd() {  
return data % 2 != 0;  
}  
@Override  
public boolean isPrime() {  
if (data <= 1) return false;  
for (int i = 2; i <= Math.sqrt(data); i++) {  
if (data % i == 0) return false;  
}  
return true;  
}  
@Override  
public long factorial() {  
int result = 1;  
if (data > 0)  
for (int i = 1; i <= data; i++) {  
result \*= i;  
}  
return result;  
}  
@Override  
public int sumOfDigits() {  
int sum = 0;  
int number = Math.abs(data);  
while (number != 0) {  
sum += number % 10;

number /= 10;  
}  
return sum;  
}  
}  
public class Main {  
public static void main(String[] args) {  
MyNumber num = new MyNumber(5);  
System.out.println("Is Positive: " + num.isPositive());  
System.out.println("Is Negative: " + num.isNegative());  
System.out.println("Is Even: " + num.isEven());  
System.out.println("Is Odd: " + num.isOdd());  
System.out.println("Is Prime: " + num.isPrime());  
System.out.println("Factorial: " + num.factorial());  
System.out.println("Sum of Digits: " + num.sumOfDigits());  
}  
}  
Output:  
Is Positive: true  
Is Negative: false  
Is Even: false  
Is Odd: true  
Is Prime: true  
Factorial: 120  
Sum of Digits: 5  
2. Define an Interface Shape with abstract method area(). Write a java  
program to calculate an area of Circle and Sphere.(use final keyword)  
interface Shape {  
final double PI = Math.PI;  
double area();  
}  
class Circle implements Shape {  
private final double radius;

public Circle(double radius) {  
this.radius = radius;  
}  
@Override  
public double  
area() {  
return PI \* radius \* radius;  
}  
}  
class Sphere implements Shape {  
private final double radius;  
public Sphere(double radius) {  
this.radius = radius;  
}  
@Override  
public double area() {  
return 4 \* PI \* radius \* radius;  
}  
}  
public class Main {  
public static void main(String[] args) {  
Circle circle = new Circle(5);  
Sphere sphere = new Sphere(3);  
System.out.println("Area of circle: " + circle.area());  
System.out.println("Area of sphere: " + sphere.area());  
}  
}  
Output:

Area of circle: 78.53981633974483  
Area of sphere: 113.09733552923255  
3. Define an interface “Operation” which has methods  
area(),volume().Define a constant PI having a value 3.142.Create a  
class cylinder which implements this interface (members – radius,  
height) Create n object and calculate the area and volume.  
interface Operation {  
double PI = 3.142;  
double area();  
double volume();  
}  
class Cylinder implements Operation {  
private final double radius;  
private final double height;  
public Cylinder(double radius, double height) {  
this.radius = radius;  
this.height = height;  
}  
@Override  
public double area() {  
return 2 \* PI \* radius \* (radius + height);  
}  
@Override  
public double volume() {  
return PI \* radius \* radius \* height;  
}  
}

public class Main {  
public static void main(String[] args) {  
Cylinder cylinder1 = new Cylinder(5, 10);  
Cylinder cylinder2 = new Cylinder(3, 7);  
System.out.println("Cylinder 1:");  
System.out.println("Area: " + cylinder1.area());  
System.out.println("Volume: " + cylinder1.volume());  
System.out.println("Cylinder 2:");  
System.out.println("Area: " + cylinder2.area());  
System.out.println("Volume: " + cylinder2.volume());  
}  
}  
Output:  
Cylinder 1:  
Area: 471.29999999999995  
Volume: 785.5  
Cylinder 2:  
Area: 188.52  
Volume: 197.946  
4. Write a program to using marker interface create a class  
product(product\_id, product\_name, product\_cost, product\_quantity)  
define a default and parameterized constructor. Create objects of class  
product and display the contents of each object  
interface Markable {  
}  
class Product implements Markable {  
private int productId;  
private String productName;  
private double productCost;  
private int productQuantity;

public Product() {  
productId = 0;  
productName = "";  
productCost = 0.0;  
productQuantity = 0;  
}  
public Product(int productId, String productName, double  
productCost, int productQuantity) {  
this.productId = productId;  
this.productName = productName;  
this.productCost = productCost;  
this.productQuantity = productQuantity;  
}  
public void displayProduct() {  
System.out.println("Product ID: " + productId);  
System.out.println("Product Name: " + productName);  
System.out.println("Product Cost: " + productCost);  
System.out.println("Product Quantity: " + productQuantity);  
}  
}  
public class Main {  
public static void main(String[] args) {  
Product product1 = new Product(202, "Smartphone", 499.99, 25);  
Product product2 = new Product(101, "Laptop", 80000.0, 5);  
product1.displayProduct();  
System.out.println();  
product2.displayProduct();

}  
}  
Output:  
Product ID: 202  
Product Name: Smartphone  
Product Cost: 499.99  
Product Quantity: 25  
Product ID: 101  
Product Name: Laptop  
Product Cost: 80000.0  
Product Quantity: 5  
5. Write a program to find the cube of given number using function  
interface.  
import java.util.Scanner;  
interface CubeCalculator {  
double calculate(double number);  
}  
public class Main {  
public static void main(String[] args) {  
Scanner scanner = new Scanner(System.in);  
System.out.print("Enter a number to find its cube: ");  
double number = scanner.nextDouble();  
CubeCalculator cubeCalculator = (n) -> n \* n \* n;  
double result = cubeCalculator.calculate(number);  
System.out.println("The cube of " + number + " is: " + result);  
}  
}  
Output:  
Enter a number to find its cube: 4  
The cube of 4.0 is: 64.0

6. Create an interface “CreditCardInterface” with methods to  
viewCreditAmount, viewPin, changePin, useCard and payBalance.  
Create a class Customer (name, card number, pin, creditAmount –  
initialized to 0). Implement methods viewCreditAmount, viewPin,  
changePin and payBalance of the interface. From Customer, create  
classes RegularCardHolder (maxCreditLimit) and GoldCardHolder  
(String specialPrivileges) and define the remaining methods of the  
interface. Create n objects of the RegularCardHolder and  
GoldCardHolder classes and write a menu driven program to perform  
the following actions 1. Use Card 2. Pay Balance 3. Change Pin  
import java.util.Scanner;  
interface CreditCardInterface {  
void viewCreditAmount();  
void viewPin();  
void changePin();  
void useCard(double amount);  
void payBalance(double amount);  
}  
abstract class Customer implements CreditCardInterface {  
String name;  
long cardNumber;  
int pin;  
double creditAmount;  
public Customer(String name, long cardNumber, int pin) {  
this.name = name;  
this.cardNumber = cardNumber;  
this.pin = pin;  
this.creditAmount = 0;  
}  
@Override  
public void viewCreditAmount() {  
System.out.println("Credit Amount: " + creditAmount);

}  
@Override  
public void viewPin() {  
System.out.println("Pin: " + pin);  
}  
@Override  
public void changePin() {  
Scanner scanner = new Scanner(System.in);  
System.out.print("Enter new pin: ");  
int newPin = scanner.nextInt();  
pin = newPin;  
System.out.println("Pin changed successfully");  
}  
@Override  
public void payBalance(double amount) {  
if (amount > creditAmount) {  
System.out.println("Insufficient balance");  
} else {  
creditAmount -= amount;  
System.out.println("Payment successful. New balance: " +  
creditAmount);  
}  
}  
}  
class RegularCardHolder extends Customer {  
double maxCreditLimit;  
public RegularCardHolder(String name, long cardNumber, int pin,  
double maxCreditLimit) {  
super(name, cardNumber, pin);  
this.maxCreditLimit = maxCreditLimit;  
}

@Override  
public void useCard(double amount) {  
if (creditAmount + amount > maxCreditLimit) {  
System.out.println("Credit limit exceeded");  
} else {  
creditAmount += amount;  
System.out.println("Card used successfully. New balance: " +  
creditAmount);  
}  
}  
}  
class GoldCardHolder extends Customer {  
String specialPrivileges;  
public GoldCardHolder(String name, long cardNumber, int pin, String  
specialPrivileges) {  
super(name, cardNumber, pin);  
this.specialPrivileges = specialPrivileges;  
}  
@Override  
public void useCard(double amount) {  
creditAmount += amount;  
System.out.println("Card used successfully. New balance: " +  
creditAmount);  
}  
}  
public class Main {  
public static void main(String[] args) {  
Scanner scanner = new Scanner(System.in);  
int n;  
System.out.print("Enter number of customers: ");  
n = scanner.nextInt();

CreditCardInterface[] customers = new CreditCardInterface[n];  
for (int i = 0; i < n; i++) {  
System.out.println("Enter customer details:");  
System.out.print("Name: ");  
String name = scanner.next();  
System.out.print("Card number: ");  
long cardNumber = scanner.nextLong();  
System.out.print("Pin: ");  
int pin = scanner.nextInt();  
System.out.print("Card type (R for Regular, G for Gold): ");  
char cardType = scanner.next().charAt(0);  
if (cardType == 'R') {  
System.out.print("Max credit limit: ");  
double maxCreditLimit = scanner.nextDouble();  
customers[i] = new RegularCardHolder(name, cardNumber, pin,  
maxCreditLimit);  
} else if (cardType == 'G') {  
System.out.print("Special privileges: ");  
String specialPrivileges = scanner.next();  
customers[i] = new GoldCardHolder(name, cardNumber, pin,  
specialPrivileges);  
} else {  
System.out.println("Invalid card type");  
i--;  
}  
}  
int choice;  
do {  
System.out.println("1. Use Card");  
System.out.println("2. Pay Balance");  
System.out.println("3. Change Pin");  
System.out.println("4. View Credit Amount");

System.out.println("5. Exit");  
System.out.print("Enter your choice: ");  
choice = scanner.nextInt();  
if (choice >= 1 && choice <= 5) {  
System.out.print("Enter customer index: ");  
int index = scanner.nextInt();  
if (index >= 0 && index < n) {  
CreditCardInterface customer = customers[index];  
switch (choice) {  
case 1:  
System.out.print("Enter amount: ");  
double amount = scanner.nextDouble();  
customer.useCard(amount);  
break;  
case 2:  
System.out.print("Enter amount: ");  
amount = scanner.nextDouble();  
customer.payBalance(amount);  
break;  
case 3:  
customer.changePin();  
break;  
case 4:  
customer.viewCreditAmount();  
break;  
}  
} else {  
System.out.println("Invalid customer index");  
}  
} else {  
System.out.println("Invalid choice");  
}  
} while (choice != 5);  
}  
}

Output:  
Enter number of customers: 2  
Enter customer details:  
Name: vaishu  
Card number: 12  
Pin: 1234  
Card type (R for Regular, G for Gold): R  
Max credit limit: 40000  
Enter customer details:  
Name: veera  
Card number: 13  
Pin: 1234  
Card type (R for Regular, G for Gold): G  
Special privileges: yes  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit  
Enter your choice: 2  
Enter customer index: 1  
Enter amount: 2000  
Insufficient balance  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit  
Enter your choice: 3  
Enter customer index: 2  
Invalid customer index  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit

Enter your choice: 3  
Enter customer index: 1  
Enter new pin: 12345  
Pin changed successfully  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit  
Enter your choice: 4  
Enter customer index: 1  
Credit Amount: 0.0  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit  
Enter your choice: 4  
Enter customer index: 2  
Invalid customer index  
1. Use Card  
2. Pay Balance  
3. Change Pin  
4. View Credit Amount  
5. Exit  
Enter your choice:  
7. Define an interface “StackOperations” which declares methods for a  
static stack. Define a class “MyStack” which contains an array and top  
as data members and implements the above interface. Initialize the  
stack using a constructor. Write a menu driven program to perform  
operations on a stack object.  
interface StackOperations {  
void push(int data);

int pop();  
int peek();  
boolean isEmpty();  
boolean isFull();  
}  
class MyStack implements StackOperations {  
private int[] stack;  
private int top;  
private final int MAX\_SIZE = 100;  
public MyStack() {  
stack = new int[MAX\_SIZE];  
top = -1;  
}  
@Override  
public void push(int data) {  
if (isFull()) {  
System.out.println("Stack Overflow");  
return;  
}  
top++;  
stack[top] = data;  
}  
@Override  
public int pop() {  
if (isEmpty()) {  
System.out.println("Stack Underflow");  
return -1;  
}  
int data = stack[top];  
top--;  
return data;  
}

@Override  
public int peek() {  
if (isEmpty()) {  
System.out.println("Stack is empty");  
return -1;  
}  
return stack[top];  
}  
@Override  
public boolean isEmpty() {  
return top == -1;  
}  
@Override  
public boolean isFull() {  
return top == MAX\_SIZE - 1;  
}  
}  
public class Main {  
public static void main(String[] args) {  
MyStack stack = new MyStack();  
int choice;  
do {  
System.out.println("1. Push");  
System.out.println("2. Pop");  
System.out.println("3. Peek");  
System.out.println("4. Check Empty");  
System.out.println("5. Check Full");  
System.out.println("6. Exit");  
System.out.print("Enter your choice: ");  
choice = Integer.parseInt(System.console().readLine());

switch (choice) {  
case 1:  
System.out.print("Enter data to push: ");  
int data = Integer.parseInt(System.console().readLine());  
stack.push(data);  
break;  
case 2:  
int popped = stack.pop();  
if (popped != -1) {  
System.out.println("Popped element: " + popped);  
}  
break;  
case 3:  
int peeked = stack.peek();  
if (peeked != -1) {  
System.out.println("Top element: " + peeked);  
}  
break;  
case 4:  
if (stack.isEmpty()) {  
System.out.println("Stack is empty");  
} else {  
System.out.println("Stack is not empty");  
}  
break;  
case 5:  
if (stack.isFull()) {  
System.out.println("Stack is full");  
} else {  
System.out.println("Stack is not full");  
}  
break;  
case 6:  
System.out.println("Exiting...");  
break;

default:  
System.out.println("Invalid choice");  
}  
} while (choice != 6);  
}  
}  
Output:  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 1  
Enter data to push: 1234  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 3  
Top element: 1234  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 4  
Stack is not empty  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full

6. Exit  
Enter your choice: 5  
Stack is not full  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 2  
Popped element: 1234  
1. Push  
2. Pop  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 6  
Exiting...  
8. Define an interface “QueueOperations” which declares methods for a  
static queue. Define a class “MyQueue” which contains an array and front  
and rear as data members and implements the above interface. Initialize  
the queue using a constructor. Write a menu driven program to perform  
operations on a queue object.  
interface QueueOperations {  
void enqueue(int data);  
int dequeue();  
int peek();  
boolean isEmpty();  
boolean isFull();  
}

class MyQueue implements QueueOperations {  
private int[] queue;  
private int front, rear;  
private final int MAX\_SIZE = 100;  
public MyQueue() {  
queue = new int[MAX\_SIZE];  
front = -1;  
rear = -1;  
}  
@Override  
public void enqueue(int data) {  
if (isFull()) {  
System.out.println("Queue Overflow");  
return;  
}  
if (isEmpty()) {  
front = 0;  
}  
rear++;  
queue[rear] = data;  
}  
@Override  
public int dequeue() {  
if (isEmpty()) {  
System.out.println("Queue Underflow");  
return -1;  
}  
int data = queue[front];  
if (front == rear) {  
front = rear = -1;  
} else {  
front++;  
}

return data;  
}  
@Override  
public int peek() {  
if (isEmpty()) {  
System.out.println("Queue is empty");  
return -1;  
}  
return queue[front];  
}  
@Override  
public boolean isEmpty() {  
return front == -1;  
}  
@Override  
public boolean isFull() {  
return rear == MAX\_SIZE - 1;  
}  
}  
public class Main {  
public static void main(String[] args) {  
MyQueue queue = new MyQueue();  
int choice;  
do {  
System.out.println("1. Enqueue");  
System.out.println("2. Dequeue");  
System.out.println("3. Peek");  
System.out.println("4. Check Empty");  
System.out.println("5. Check Full");  
System.out.println("6. Exit");  
System.out.print("Enter your choice: ");

choice = Integer.parseInt(System.console().readLine());  
switch (choice) {  
case 1:  
System.out.print("Enter data to enqueue: ");  
int data = Integer.parseInt(System.console().readLine());  
queue.enqueue(data);  
break;  
case 2:  
int dequeued = queue.dequeue();  
if (dequeued != -1) {  
System.out.println("Dequeued element: " + dequeued);  
}  
break;  
case 3:  
int peeked = queue.peek();  
if (peeked != -1) {  
System.out.println("Front element: " + peeked);  
}  
break;  
case 4:  
if (queue.isEmpty()) {  
System.out.println("Queue is empty");  
} else {  
System.out.println("Queue is not empty");  
}  
break;  
case 5:  
if (queue.isFull()) {  
System.out.println("Queue is full");  
} else {  
System.out.println("Queue is not full");  
}  
break;  
case 6:

System.out.println("Exiting...");  
break;  
default:  
System.out.println("Invalid choice");  
}  
} while (choice != 6);  
}  
}  
Output:  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 1  
Enter data to enqueue: 123456  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 3  
Front element: 123456  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 4  
Queue is not empty  
1. Enqueue  
2. Dequeue  
3. Peek

4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 2  
Dequeued element: 123456  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 5  
Queue is not full  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Check Empty  
5. Check Full  
6. Exit  
Enter your choice: 6  
Exiting...