1. Design a Java interface for ADT Stack with the following methods.
Maximum size of stack=15
boolean isFull()
boolean isEmpty()
void push(int element)
int pop()
int peep()
Implement this interface using a class with an integer array to store the elements to perform the
following stack operations.
Boolean balanceParanthesis(String expression)
boolean checkTwoStacks(Stack s1, Stack s2)

```
import java.util.*;
interface StackADT
{
int MAXSIZE = 15;
boolean isFull();
boolean isEmpty();
void push(int element);
int pop();
int peek();
}
class Stack implements StackADT
{
private int stack[] = new int[MAXSIZE];
private int top = -1;
public boolean isFull()
{
if(top == MAXSIZE-1)
return true;
else
return false;
}
public boolean isEmpty()
{
if(top == -1)
return true;
else

return false;
}
public void push(int element)
{
if(!isFull())
stack[++top] = element;
else
```

```
System.out.print("\nStack is Full! Can\'t push element...");
}
public int pop()
{
if(!isEmpty())
return stack[top--];
else
return -1;
}
public int peek()
{
if(!isEmpty())
return stack[top];
else
return -1;
}
boolean balanceParanthesis(String expression)
{
int len = expression.length();
for(int i=0; i<len; i++)
{
if(expression.charAt(i) == '(' || expression.charAt(i) == '{' ||
expression.charAt(i) == '[')
push(expression.charAt(i));
else
{
switch(expression.charAt(i))
{
case ')':
if(!isEmpty() && peek() == '(')
pop();
else
return false;
break;
case '}':
if(!isEmpty() && peek() == '{')
pop();
else
return false;
break;
case ']':
if(!isEmpty() && peek() == '[')
pop();
else
return false;

break;
}
}
}
if(isEmpty())
return true;
else
return false;
}
int length()
{
return top+1;
}
boolean isIn(Stack s)
{
while(!s.isEmpty())
{
int num = s.peek();
for(int i=top; i>=0; i--)
{
if(num == stack[i])
break;
else if(i == 0)
return false;
}
s.pop();
}
return true;
}
void display()
{
for(int i=top; i>0; i--)
System.out.print(stack[i]+"<--");
System.out.println(stack[0]);
}
}
class StackProgram
{
static void copyStack(Stack dest, Stack src)
{
int l = src.length();
int arr[] = new int[l];
for(int i=0; i<l; i++)
arr[i] = src.pop();
for(int i=l-1; i>=0; i--)
{
src.push(arr[i]);
dest.push(arr[i]);
```

```
}
}
static boolean checkTwoStacks(Stack s1, Stack s2)

{
Stack temp1 = new Stack();
Stack temp2 = new Stack();
copyStack(temp1, s1);
copyStack(temp2, s2);
while(!temp1.isEmpty() && !temp2.isEmpty())
{
if(temp1.peek() == temp2.peek())
{
temp1.pop();
temp2.pop();
}
else
break;
}
if(temp1.isEmpty() && temp2.isEmpty())
{
System.out.println("Same order.");
return true;
}
else if(s1.length() == s2.length())
{
boolean flag1 = false, flag2 = false;
Stack temp = new Stack();
copyStack(temp, s2);
flag1 = s1.isIn(temp);
flag2 = s2.isIn(s1);
if(flag1 && flag2)
{
System.out.println("Same elements,Different orders.");
return true;
}
return false;
}
return false;
}
public static void main(String args[])
{
Scanner s = new Scanner(System.in);
int choice;
int control =1;
while(control==1)
{
```

```
System.out.println("Enter 1 for balance the expression and 2 for
checking the
stacks...");
choice=s.nextInt();
s.nextLine();
switch(choice)
{
case 1:
System.out.print("\nEnter the expression : ");
String exp = s.nextLine();
s.nextLine();
Stack st = new Stack();

if(st.balanceParanthesis(exp))
System.out.println("Balanced.");
else
System.out.println("Not Balanced.");
break;
case 2:
Stack s1 = new Stack();
Stack s2 = new Stack();
System.out.println("1. Push to Stack 1\n2. Push to Stack 2\n3.
Pop
from Stack 1\n4. Pop from Stack 2\n5. Check the two Stacks\n");
while(true)
{
System.out.print("\nYour Choice : ");
int temp = s.nextInt();
if(temp == 5)
break;
int n;
switch(temp)
{
case 1:
System.out.print("Enter the number : ");
n = s.nextInt();
s1.push(n);
break;
case 2:
System.out.print("Enter the number : ");
n = s.nextInt();
s2.push(n);
break;
case 3:
s1.pop();
break;
case 4:
```

```
s2.pop();
break;
default :
System.out.println("ERROR...Enter
again.");
}
}
System.out.println("Stack 1 : ");
s1.display();
System.out.println("Stack 2 : ");
s2.display();
if(checkTwoStacks(s1, s2))
System.out.println("Same");
else
System.out.println("Different");

break;
}
System.out.print("To continue press 1... To stop press 2...");
control = s.nextInt();
}
}
}

/*
OUTPUT:
Enter 1 for balance the expression and 2 for checking the
stacks...
1
Enter the expression : ({}{}][])]
Not Balanced.
To continue press 1... To stop press 2...1
Enter 1 for balance the expression and 2 for checking the
stacks...
2
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Check the two Stacks

Your Choice : 1
Enter the number : 23
Your Choice : 1
Enter the number : 34
Your Choice : 1
Enter the number : 45
```

```
Your Choice : 2
Enter the number : 23
Your Choice : 2
Enter the number : 45
Your Choice : 2
Enter the number : 34
Your Choice : 5
Stack 1 :
45<--34<--23
Stack 2 :
34<--45<--23

Same elements,Different orders.
Same
To continue press 1... To stop press 2...1
Enter 1 for balance the expression and 2 for checking the
stacks...
2
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Check the two Stacks

Your Choice : 1
Enter the number : 23
Your Choice : 1
Enter the number : 34
Your Choice : 1
Enter the number : 45
Your Choice : 2
Enter the number : 23
Your Choice : 2
Enter the number : 34
Your Choice : 2
Enter the number : 45
Your Choice : 5
Stack 1 :
45<--34<--23
Stack 2 :
45<--34<--23
Same order.
Same
To continue press 1... To stop press 2...1
Enter 1 for balance the expression and 2 for checking the
stacks...
2
```

```
1. Push to Stack 1
2. Push to Stack 2
3. Pop from Stack 1
4. Pop from Stack 2
5. Check the two Stacks

Your Choice : 1
Enter the number : 23
Your Choice : 1
Enter the number : 34
Your Choice : 1
Enter the number : 45
Your Choice : 2

Enter the number : 32
Your Choice : 2
Enter the number : 43
Your Choice : 2
Enter the number : 54
Your Choice : 5
Stack 1 :
45<--34<--23
Stack 2 :
54<--43<--32
Different
To continue press 1... To stop press 2...2
*/
```

2. Create an interface named **which consists of the following methods**
a. getName()
b. setName(String)
c. getMaxPassengers()
d. setMaxPassengers(int)
e. getMaxSpeed()
f. setMaxSpeed(int)
Inherit an interface named from which contains the following methods
a) getNumWheels()
b) setNumWheels(int)
c) drive()
Inherit an interface named from which contains the following methods
a) getDisplacement()
b) setDisplacement(int)
c) launch()Create a class inherits from which has soundHorn() method.

Create a class inherits from which has fireGun() method.
Create a class inherits from both and which
has enterLand() and enterSea() methods.
Create an Interface which contains soundSiren() method.
Create a class inherits from and
which has patientIn() method.
Draw the class diagram which shows the above problem and
implement it.

```
import java.util.*;
interface Vehicle
{

public String getName();
public void setName(String name);
public int getMaxPassengers();
public void setMaxPassengers(int maxPass);
public int getMaxSpeed();
public void setMaxSpeed(int maxSpeed);
}
interface LandVehicle extends Vehicle
{
public int getNumWheels();
public void setNumWheels(int numWheels);
public void drive();
}
interface SeaVehicle extends Vehicle
{
public int getDisplacement();
public void setDisplacement(int disp);
public void launch();
}
interface EmergencyVehicle
{
public void soundSiren();
}
class Jeep implements LandVehicle
{
private String name;
private int maxPass, maxSpeed, numWheels;
public String getName()
{
return name;
}
public void setName(String name)
{
this.name = name;
}
public int getMaxPassengers()
{
return maxPass;
}
public void setMaxPassengers(int maxPass)
{
this.maxPass = maxPass;
}
public int getMaxSpeed()
{
return maxSpeed;
}
public void setMaxSpeed(int maxSpeed)
{
this.maxSpeed = maxSpeed;
}
public int getNumWheels()

{
return numWheels;
}
public void setNumWheels(int numWheels)
{
this.numWheels = numWheels;
}
public void drive()
{
System.out.println("Vehicle Started...");
}
public void soundHorn()
{
System.out.println("Beep Beep");
}
}
class Frigate implements SeaVehicle
{
private String name;
private int maxPass, maxSpeed, disp;
public String getName()
{
return name;
}
public void setName(String name)
{
this.name = name;
}
```

```
public int getMaxPassengers()
{
return maxPass;
}
public void setMaxPassengers(int maxPass)
{
this.maxPass = maxPass;
}
public int getMaxSpeed()
{
return maxSpeed;
}
public void setMaxSpeed(int maxSpeed)
{
this.maxSpeed = maxSpeed;
}
public int getDisplacement()
{
return disp;
}
public void setDisplacement(int disp)
{
this.disp = disp;
}
public void launch()
{
System.out.println("Launched the Frigate...");

}
public void fireGun()
{
System.out.println("Launched the torpedos...");
}
}
class HoverCraft implements LandVehicle, SeaVehicle
{
private String name;
private int maxPass, maxSpeed, numWheels, disp;
public String getName()
{
return name;
}
public void setName(String name)
{
this.name = name;
}
public int getMaxPassengers()
```

```
{
return maxPass;
}
public void setMaxPassengers(int maxPass)
{
this.maxPass = maxPass;
}
public int getMaxSpeed()
{
return maxSpeed;
}
public void setMaxSpeed(int maxSpeed)
{
this.maxSpeed = maxSpeed;
}
public int getNumWheels()
{
return numWheels;
}
public void setNumWheels(int numWheels)
{
this.numWheels = numWheels;
}
public void drive()
{
System.out.println("Launched the land mode...");
}
public int getDisplacement()
{
return disp;
}
public void setDisplacement(int disp)
{
this.disp = disp;
}
public void launch()

{
System.out.println("Launched the Frigate mode...");
}
public void enterLand()
{
drive();
System.out.println("land mode...");
}
public void enterSea()
{
```

```
launch();
System.out.println("Sea mode...");
}
}
class Ambulance implements LandVehicle, EmergencyVehicle
{
private String name;
private int maxPass, maxSpeed, numWheels;
public String getName()
{
return name;
}
public void setName(String name)
{
this.name = name;
}
public int getMaxPassengers()
{
return maxPass;
}
public void setMaxPassengers(int maxPass)
{
this.maxPass = maxPass;
}
public int getMaxSpeed()
{
return maxSpeed;
}
public void setMaxSpeed(int maxSpeed)
{
this.maxSpeed = maxSpeed;
}
public int getNumWheels()
{
return numWheels;
}
public void setNumWheels(int numWheels)
{
this.numWheels = numWheels;
}
public void drive()
{
System.out.println("Launched the Ambulance...");
}

public void soundSiren()
{
```

```
System.out.println("Mua Mua Mua ... ");
}
public void patientIn()
{
drive();
soundSiren();
System.out.println("Fetched the patients...");
}
}
class Main
{
public static void main(String args[])
{
String choice;
int cont=1;
Scanner s=new Scanner(System.in);
while(cont==1)
{
System.out.println("Enter jeep or Frigate or Ambulance or
HoverCraft...");
choice=s.nextLine();
switch(choice)
{
case "jeep":
Jeep obj1=new Jeep();
obj1.drive();
break;
case "frigate":
Frigate obj2 =new Frigate();
obj2.launch();
break;
case "ambulance":
Ambulance obj3 =new Ambulance();
obj3.patientIn();
break;
case "hovercraft":
HoverCraft obj4=new HoverCraft();
System.out.println("Enter 1 for land and 2 for sea...");
int x=s.nextInt();
s.nextLine();
if(x==1)
obj4.enterLand();
else
obj4.enterSea();
break;
}
System.out.println("Enter 1 to continue or 2 to stop...");
cont=s.nextInt();
s.nextLine();
}
}
}

/*
OUTPUT:
Enter jeep or Frigate or Ambulance or HoverCraft...
hovercraft
Enter 1 for land and 2 for sea...
1
Launched the land mode...
land mode...
Enter 1 to continue or 2 to stop...
1
Enter jeep or Frigate or Ambulance or HoverCraft...
hovercraft
Enter 1 for land and 2 for sea...
2
Launched the Frigate mode...
Sea mode...
Enter 1 to continue or 2 to stop...
2
*/
```