Day-10 JA-111 Batch

Topics

- **Interface**
- **inheritance & interfaces**
- **default & private method in interfaces**
- **static method in interfaces**
- **Comparison between abstract class & interfaces**

interface

- An interface is used to define a generic template which can be implemented by various classes such that classes may be related or unrelated.
- All methods in an interface are abstract such that no need to use abstract keyword with methods of interfaces
- All variables in an interface are static and final so mandatory to provide the value [No instance variables and constructor in interface]
- To define an interface

```
[access-modifier] interface interface-name{
    list of constants and abstract methods
}
```

- All the methods of interface are considered public by default (no matter it is specified or not) so while overriding methods of interface it is mandatory to use public keyword because in overridding weaker access cannot be used.
- To implement an interface 'implements' keyword is used. An interface can be implemented by any number of classes and one class can implement any number of interfaces.

```
[access-modifier] class class-name [extends super-class] implements
interface-name-1[,interface-name-2[,interface-name-3]]{
    body of class
}
```

- The implementing class of an interface must provide definiton of all the abstract methods of interface, otherwise it will also be an abstract class.
- It is okay to create reference variable of an interface but it is not possible to instantiate an interface. The reference variable of interface can point to the object of implementing class.
- Using the reference variable of interface, when overridden method is called then calling decision took place on the behalf of the object being pointed by the reference variables which is Dynamic Method Dispatch.
- The interface also represents is-a relationship. It achieves total abstraction.

We Activity

Point out error if any in the code

```
interface A{
  float a = 5.7f;
  void show();
}

class B implements A{
  void get(){
    System.out.println("inside get");
    a = 5.8f;
  }
}

class C implements A{
  void show(){
    System.out.println("inside show");
  }

  void foo(){
    System.out.println("inside foo");
  }
}

class Demo{
  public static void main(String args[]){
    A a = new A();
    A a1 = new C();
    a1.show();
    a1.foo();

    C c1 = new C();
    c1.foo();
  }
}
```

Solution

```
interface A{
  float a = 5.7f;
  void show();
}
//either class B should be abstract or class B must provide definition
of public void show
abstract class B implements A{
  void get(){
    System.out.println("inside get");
    //a = 5.8f;  a is final variable so value cannot be changed.
  }
}

class C implements A{
  //while implementing method of interface, make sure to use public
access modifier
  public void show(){
    System.out.println("inside show");
  }

  void foo(){
    System.out.println("inside foo");
  }
}

class Demo{
  public static void main(String args[]){
    //A a = new A();  Not possible to create object of interface
    A a1 = new C();
    a1.show();
    //a1.foo(); using ref var of interface you can access members
written in interface only

    C c1 = new C();
    c1.foo();
  }
}
```

An Example

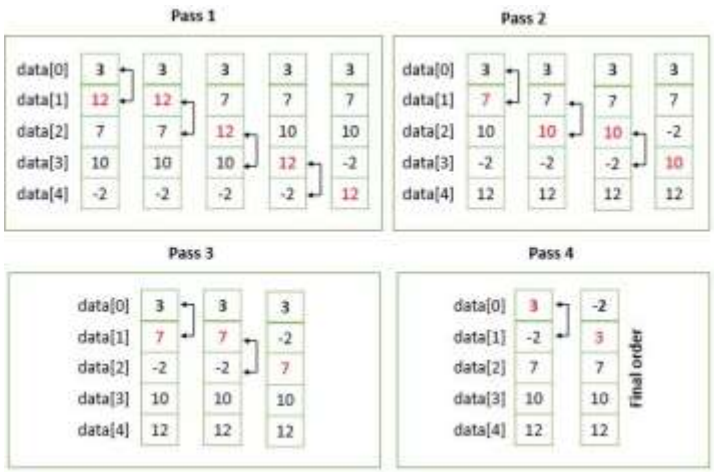
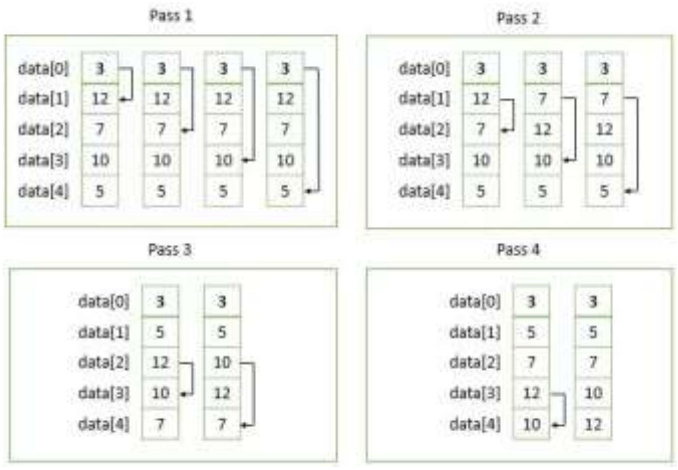
```
package com.masai.sprint3;

public interface Mensuration{
  double PI = 3.14;
  public double getVolume();
  public double getSurfaceArea();
}

package com.masai.sprint3;

public class Sphere implements Mensuration{
  private double radius;

  public Sphere(double radius) {
    this.radius = radius;
  }

  @Override
  public double getVolume() {
    double volume = 4.0/3.0 * PI * Math.pow(radius, 3);
    return Math.round(volume * 100)/100.0;
  }

  @Override
  public double getSurfaceArea() {
    double surfaceArea = 4 * PI * Math.pow(radius, 2);
    return Math.round(surfaceArea * 100)/100.0;
  }
}
```

```
package com.masai.sprint3;

public class Cone implements Mensuration {
  private double radius;
  private double height;
  private double slantHeight;

  public Cone(double radius, double height) {
    this.radius = radius;
    this.height = height;
    this.slantHeight = Math.sqrt(Math.pow(radius, 2) + Math.pow(height,
2));
  }

  @Override
  public double getVolume() {
    double volume = 1.0/3.0 * PI * Math.pow(radius, 2) * height;
    return Math.round(volume * 100)/100.0;
  }

  @Override
  public double getSurfaceArea() {
    double surfaceArea = PI * radius * (radius + slantHeight);
    return Math.round(surfaceArea * 100)/100.0;
  }
}

package com.masai.sprint3;

public class MensurationDemo {
  public static void main(String[] args) {
    Mensuration m = null;

    m = new Sphere(7.0);
    System.out.println("The surface area of sphere is " +
m.getSurfaceArea());
    System.out.println("The volume of sphere is " + m.getVolume());

    m = new Cone(5.0, 12.0);
    System.out.println("The surface area of cone is " +
m.getSurfaceArea());
    System.out.println("The volume of cone is " + m.getVolume());
  }
}
```

Output

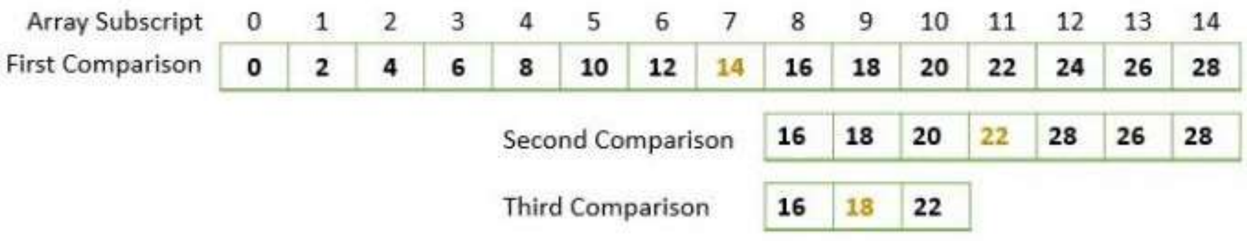
```
The surface area of sphere is 615.44
The volume of sphere is 1436.03
The surface area of cone is 282.6
The volume of cone is 314.0
```

Before going to next example, take a quick revision of bubble sort, selection sort and binary search

An example of bubble sort	An example of selection sort
 Pass 1 Pass 2 Pass 3 Pass 4 data[0] data[1] data[2] data[3] data[4] Final order	 Pass 1 Pass 2 Pass 3 Pass 4 data[0] data[1] data[2] data[3] data[4]

An example of binary search

Search for element 18 in the array


Array Subscript 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
First Comparison 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28
Second Comparison 16 18 20 22 28 26 28
Third Comparison 16 18 22

At third comparison the search value equal to middle value and hence return position 9

An Example

```
package com.masai.sprint3;
public interface SortArrayElements {
  void sort(int arr[]);
}

package com.masai.sprint3;
public class BubbleSort implements SortArrayElements {
  @Override
  public void sort(int arr[]) {
    for(int i = 0; i < arr.length - 1; i++) {
      for(int j = 0; j < (arr.length - 1 - i); j++) {
        if(arr[j] > arr[j + 1]) {
          int temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }
  }
```

```java
}

package com.masai.sprint3;
public class SelectionSort implements SortArrayElements {
  private int getMinIndex(int arr[], int startIndex) {
    int min = startIndex;
    for(int i = startIndex + 1; i < arr.length; i++) {
      if(arr[min] > arr[i])
        min = i;
    }
    return min;
  }

  @Override
  public void sort(int arr[]) {
    for(int i = 0; i < arr.length; i++) {
      int minIndex = getMinIndex(arr, i);
      int temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
    }
  }
}

package com.masai.sprint3;
public class SortArrayElementsMain {
  public static void main(String args[]) {
    int arr[] = new int[]{74, 16, 85, 73, 79, 15, 99};
    SortArrayElements sat = new BubbleSort();
    sat.sort(arr);
    for(int temp: arr){
      System.out.print(temp + " ");
    }
    System.out.println();

    System.out.println("-=====================");

    int brr[] = new int[]{45, 17, 86, 72, 78, 16, 98};
    sat = new SelectionSort();
    sat.sort(brr);
    for(int temp: brr){
      System.out.print(temp + " ");
    }
    System.out.println();
  }
}
```
Output
```
15 16 73 74 79 85 99
-=====================
16 17 45 72 78 86 98
```

As we can see that sort method has different implementations in both classes say we want to integrate one more functionality that is searching in the elements of the array, to do the same we have to write method signature in the interface but have to write implementation for the same in both classes which leads to duplicity of the code.

Solution: use default method in interface and use private methods also to create small helper methods (if required)

- An interface can have method with body such that for such methods default keyword has to be used with the method signature. The default method of the interface can be used by the implementing class directly or it can be overridden (if required)
- An interface can have private method with body such that they are accessible in the interface only so they are used to create some supporting small methods for default methods

Add following in the interface SortArrayElements

```
default int binarySearch(int arr[], int startIndex, int endIndex, int
element) {
  if(isSorted(arr)) {
    while(startIndex <= endIndex) {
      int mid = (startIndex + endIndex)/2;
      if(arr[mid] == element) {
        return mid;
      }else if(arr[mid] > element) {
        endIndex = mid - 1;
      }
      startIndex = mid + 1;
    }
    return -1;
  }
  return -2;
}

private boolean isSorted(int arr[]) {
  for(int i = 0; i < arr.length - 1; i++) {
    if(arr[i] > arr[i + 1])
      return false;
  }
  return true;
}
```

& add following lines in class SortArrayElementsMain

```
int searchIndex = sat.binarySearch(arr, 0, arr.length - 1, 74);
System.out.println(searchIndex >= 0? "found":(searchIndex == -1?"Not
found": "Unsorted Array"));
```

```
searchIndex = sat.binarySearch(brr, 0, brr.length - 1, 74);
System.out.println(searchIndex >= 0? "found":(searchIndex == -1?"Not
found": "Unsorted Array"));
```

Output

```
found
Not found
```

The code to print the content of the array will get duplicated for every implementation, better to create a common method to return the elements of the array in the string format. We do not let subclass to override this method.

Solution: use static method in interface.

- The interface can have static methods also such that the static methods are accessible using interface name only they are not accessible using reference of interface or in implementing class directly.

Add this method in interface SortArrayElements

```
static String getArrayElementsForPrinting(int arr[]) {
  StringBuilder sb= new StringBuilder(50);
  for(int temp: arr) {
    sb.append(temp + " ");
  }
  return new String(sb.substring(0, sb.length() - 1));
}
```

& replace the for loop to print the content with the following

```
System.out.println(SortArrayElements.getArrayElementsForPrinting(arr));
System.out.println(SortArrayElements.getArrayElementsForPrinting(brr));
```

Output

```
15 16 73 74 79 85 99
-=-=-=-=-=-=-=-=-=-=-=-=-=
16 17 45 72 78 86 98
```

interface and inheritance

- interfaces can also inherited one another such that for inheritance same extends keyword is going to be used.
  ```
  interface    interface-name    extends    super-interface-name[,super-
  interface-name[,super-interface-name]]{
      body of interface
  }
  ```
- when a class implements an interface then it must provide implementation of all abstract methods of implemented interface and its super interfaces.
- In classes multiple inheritance is not supported i.e. one class cannot have more than one parent but in interfaces multiple inheritance is supported i.e. one interface can inherit more

than one interface.

An Example
```
interface Constants{
  boolean YES = true;
  boolean NO = false;
}

interface CheckEqual{
  boolean isEquals(int a, int b);
}

interface CheckSmaller{
  boolean isSmaller(int a, int b);
}

interface CheckGreater extends Constants, CheckEqual, CheckSmaller{
  boolean isGreater(int a, int b);
}

class Relational implements CheckGreater{
  public boolean isEquals(int a, int b){
    return (a == b?YES:NO);
  }

  public boolean isSmaller(int a, int b){
    return (a < b?YES:NO);
  }

  public boolean isGreater(int a, int b){
    return (a > b?YES:NO);
  }

  public static void main(String args[]){
    CheckGreater cg = new Relational();
    System.out.println("cg.isSmaller(10, 20)? " + cg.isSmaller(10, 20));
    System.out.println("cg.isEquals(10, 20)? " + cg.isEquals(10, 20));
    System.out.println("cg.isGreater(10, 20)? " + cg.isGreater(10, 20));
  }
}
```

Output
```
cg.isSmaller(10, 20)? true
cg.isEquals(10, 20)? false
cg.isGreater(10, 20)? false
```

Difference between abstract class and interface

Abstract class	Interface

Can have instance variable; can have constructors	No instance variables; no constructor
It cannot support multiple inheritance	It supports multiple inheritance.
all members function (static + non-static) are inherited	static methods are not available in the implementing class (They can be accessed using interface only)
you can use any access modifier with members	All methods are public here
Used for common base class implementation to the derived classes	Used for common contract for related or unrelated classes
Used for related class so it has high cohesion, tight coupling	Can be used for related as well as unrelated classes so it has high cohesion, loose coupling

A note on cohesion and coupling

Cohesion: Interdependency among the modules; required to be high
Coupling: Effect of change of code in module on another module; required to be low

Say we have an abstract class 'A' such that this is inherited by four different classes

```
abstract class A{
  void abc();
  void pqr();
}
class B extends A{}
class C extends A{}
class D extends A{}
class E extends A{}
```

All four classes must provide implementation of abc() and pqr() method. Say we want to add some common method xyz() for class B and D the only way to achieve this is by writing method in class A then this method has to be implemented unnecessary by class C and E also It shows that coupling among the code is high because one change is affecting other code also.

The same situation can be handled better using interfaces

```
interface A{
  void abc();
  void pqr();
}

class B implements A{}
class C implements A{}
class D implements A{}
class E implements A{}
```

All four classes must provide implementation of abc() and pqr() method. Say we want to add some common method xyz() for class B and D then better way to achieve this is by writing method in new interface say 'Z' then this interface has to be implemented by class B and D only such that class C and E remain unaffected. It shows that coupling among the code is low because one change is affecting code necessily only.

Marker/tagged interface

- An interface that has no member is known as a marker or tagged interface.
- They are used to provide some essential information to the JVM so that JVM may perform some useful operations.
  e.g. Serializable, Cloneable

You Activity

- Create an interface named Taxation which has only one method that is double calculateTax()
- You have to create two implementing classes that are SalariedPeople and BusinessPeople
- Override calculateTax() method in SalariedPeople class such that
  - if annual income is up to 2,50,000 then no taxif annual income is up to 5,00,000 then 10% tax of income of this slab
  - if annual income is up to 10,00,000 then 20% tax of taxable income tax of income of this slab
  - if annual income is more than 10,00,000 then 30% tax of taxable income tax of income of this slab
  - if annual income is more than 1,00,00,000 then 7,50,000/- is fixed surcharge

- Override calculateTax() method in BusinessPeople class such that the net profit is total sale - (total purchase + operating expenses)
  - If net profit is up to 5,00,000 then no tax
  - If net profit is up to 15,00,000 then 10% tax of net profit of this slab
  - If net profit is up to 50,00,000 then 22% tax of net profit of this slab
  - If net profit is more than 50,00,000 then 32% tax of net profit of this slab
  - If net profit is more than 5,00,00,000 then then 12,50,000/- is fixed surcharge

```
interface Taxation{
     //write code here
}

class SalariedPeople implements Taxation{
     //write code here
}

class BusinessPeople implements Taxation{
     //write code here
}

class Tester{
  public static void main(String args[]) {
    Taxation taxation = null;
    taxation = new SalariedPeople(1500000);
          System.out.println("The total tax for this salaries person is
" + taxation.calculateTax());

          taxation = new BusinessPeople(27500000, 15000000, 5100000);
          System.out.println("The total tax for this business person is
" + taxation.calculateTax());
```

```
  }
}
```

Output

```
The total tax for this salaries person is 275000/-
The total tax for this business person is 848000
```

Explanation

```
For annual income of salary 1500000
Tax for income that is above 1000000 is 150000
Tax for income that is above 500000 is 100000
Tax for income that is above 250000 is 25000
Total tax: 2,75,000
```

```
For annual income of business person with total sale: 2,75,00,000 total
purchase: 1,75,00,000 and operating cost: 51,00,000
Net profit is: 49,00,000
Tax for Net profit that is above 15,00,000 is 7,48,000
Tax for Net profit that is above 5,00,000 is 1,00,000
Total tax: 8,48,000
```