Unit 2 : Array, Inheritance and Interface

Que1 : Explain this keyword in java

- The this keyword refers to the current object in a method or constructor.
- The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

Example

- If you omit the keyword in the example below, the output would be "0" instead of "5".

```
public class Main
{
 int x;

 // Constructor with a parameter
 public Main(int x) {
  this.x = x;
 }

 // Call the constructor
 public static void main(String[] args) {
  Main myObj = new Main(5);
  System.out.println("Value of x = " + myObj.x);
 }
}
```

Que 2 : Explain Array in java

- Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

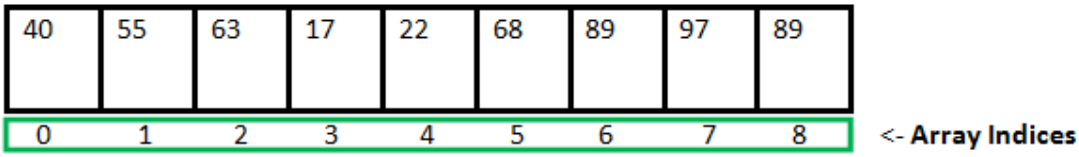
Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

There are two types of array.

- **One Dimensional Array**

  Conceptually you can think of a one-dimensional array as a row, where elements are stored one after another.



Array Length = 9
First Index = 0
Last Index = 8

Array declaration syntax in one Dimensional Array

Datataype var-name[];

Example

```
class Testarray{

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}
```

Multidimensional Array

in such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

```
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Que 3: Explain For-each Loop for Java Array

- We can also print the Java array using for-each loop.
- The Java for-each loop prints the array elements one by one.
- It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
for(data_type variable:array){
//body of the loop
}
```

Example

```
//Java Program to print the array elements using for-each loop
class Testarray1
{
public static void main(String args[])
{
int arr[]={33,3,4,5};
//printing array using for-each loop
for(int i:arr)
System.out.println(i);
}}
```

Output:

```
33
3
4
5
```

Que 4: Explain Passing arrays to methods

We can pass the java array to method so that we can reuse the same logic on any array.

```
class Testarray2{
//creating a method which receives an array as a parameter
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];

System.out.println(min);
}

public static void main(String args[]){
int a[]={33,3,4,5};//declaring and initializing an array
min(a);//passing array to method
}}
```

Output

3

Que 5 : Explain Returning Array from the Method

- Arrays can not only be passed to methods but we can use arrays as return value from methods.
- If you are faced with a situation where you want to return multiple values from a method,
- All the values can be encapsulated in an array and returned.

Example

```
//Java Program to return an array from the method

class TestReturnArray{

//creating method which returns an array

static int[] get(){

return new int[]{10,30,50,90,60};

}
```

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

//calling method which returns an array

int arr[]=get();

//printing the values of an array

for(int i=0;i<arr.length;i++)

System.out.println(arr[i]);

}}
```

Que 6 : Explain command line arguments in java

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.

Example

```
class CommandLineExample
{
public static void main(String args[])
{
System.out.println("Your first argument is: "+args[0]);
}
}
```

Output :

compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo

Que Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviours of a parent object.
- It is an important part of OOPs (Object Oriented programming system).

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
  .....
}
```

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Example

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
```

```
  Programmer p=new Programmer();
  System.out.println("Programmer salary is:"+p.salary);
  System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

OUTPUT

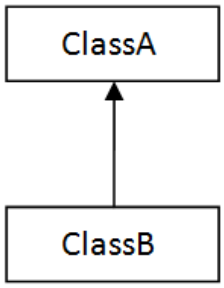
Programmer salary is:40000.0

Bonus of programmer is:10000

Types of inheritance in java

Single Inheritance

When a class inherits another class, it is known as a *single inheritance*.



Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
```

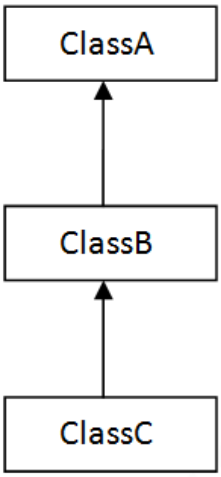
```
d.bark();

d.eat();

}}
```

OUTPUT

```
barking...
eating..
```

Multilevel Inheritance

When there is a chain of inheritance, it is known as *multilevel inheritance*



2) Multilevel

EXAMPLE

```
class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{
```

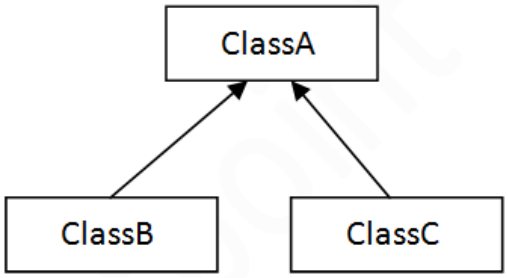
```
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating..
```

Hierarchical Inheritance

When two or more classes inherits a single class, it is known as *hierarchical inheritance*.



3) Hierarchical

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

meowing...

eating...

Multiple inheritance

- Multiple inheritance is not supported in Java through class.

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes.
- The C class inherits A and B classes.
- If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes.
- So whether you have same method or different, there will be compile time error.

Example

```
class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B{//suppose if it were


 public static void main(String args[]){

   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?

}

}
```

OUTPUT

Compile Time Error

Que : Explain Method Overriding in Java

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.

Example

```
// A Simple Java program to demonstrate
// method overriding in java

// Base Class
class Parent {
	void show()
	{
		System.out.println("Parent's show()");
	}
}

// Inherited class
class Child extends Parent {
	// This method overrides show() of Parent
	@Override
	void show()
	{
		System.out.println("Child's show()");
	}
}

// Driver class
```

```
class Main {
	public static void main(String[] args)
	{
		// If a Parent type reference refers
		// to a Parent object, then Parent's
		// show is called
		Parent obj1 = new Parent();
		obj1.show();

		// If a Parent type reference refers
		// to a Child object Child's show()
		// is called. This is called RUN TIME
		// POLYMORPHISM.
		Parent obj2 = new Child();
		obj2.show();
	}
}
```

Output:

```
Parent's show()
Child's show()
```

Que : Explain super keyword in JAVA

- The super keyword refers to superclass (parent) objects.
- It is used to call superclass methods, and to access the superclass constructor.
- The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

Example

```
class Superclass
{
  int num = 100;
}
class Subclass extends Superclass
{
  int num = 110;
  void printNumber(){
		/* Note that instead of writing num we are
		 * writing super.num in the print statement
		 * this refers to the num variable of Superclass
		 */
```

```
        System.out.println(super.num);
  }
  public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printNumber();
  }
}
```

Output:
100

Que : Explain Final keyword in java

The final keyword in java is used to restrict the user. The java final keyword can be used in many situations. Final can be:

1. **Variable**

   If you make any variable as final, you cannot change the value of final variable(It will be constant).

   Example

```
class Bike9{
  final int speedlimit=90;//final variable
  void run(){
   speedlimit=400;
  }
  public static void main(String args[]){
  Bike9 obj=new  Bike9();
  obj.run();
  }
 }//end of class
```

   OUTPUT

   Compile by: javac Bike9.java

```
26.246/Bike9.java:4: error: cannot assign a value to final variable speedlimit
  speedlimit=400;
  ^
1 error
```

2. **Method**

   If you make any method as final, you cannot override it.

   EXAMPLE

```
class Bike{
 final void run(){System.out.println("running");}
}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

   OUTPUT

```
Compile by: javac Honda.java

26.246/Honda.java:6: error: run() in Honda cannot override run() in
Bike
ly with 100kmph");}
      ^
 overridden method is final
1 error
```

3. **Class**

   If you make any class as final, you cannot extend it.

   EXAMPLE

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
```

```
}
```

OUTPUT

```
Compile by: javac Honda1.java

6.246/Honda1.java:3: error: cannot inherit from final Bike
 class Honda1 extends Bike{
                ^
6.246/Honda1.java:7: error: cannot find symbol
   Honda1 honda= new Honda();
                 ^
  symbol:   class Honda
  location: class Honda1
2 errors
```

QUE : Explain Abstract class

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message.
- You don't know the internal processing about the message delivery.

Abstract class in Java

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the body of the method..

Example

```
abstract class A{}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

Example

```
abstract class Bike{
  abstract void run();
 }

 class Honda4 extends Bike{
 void run(){System.out.println("running safely..");}

 public static void main(String args[]){
  Bike obj = new Honda4();
  obj.run();
 }
 }
```

Output

```
running safely..
```

Que : Explain Interface in java

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is a mechanism to achieve abstraction.
- There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>
{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

Java Interface Example

```
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Output

Hello

Example 2

- In this example, the Drawable interface has only one method.
- Its implementation is provided by Rectangle and Circle classes.
- In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers.
- Moreover, it is used by someone else.
- The implementation part is hidden by the user who uses the interface

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
```

```
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided
by method e.g. getDrawable()
d.draw();
}}
```

Output:

```
drawing circle
```

Example 3

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

OUTPUT

```
ROI: 9.15
```

Variables in Interface

The variable in an interface is public, static, and final by default.

- If any variable in an interface is defined without public, static, and final keywords then, the compiler automatically adds the same.
- No access modifier is allowed except the public for interface variables.
- Every variable of an interface must be initialized in the interface itself.
- The class that implements an interface can not modify the interface variable, but it may use as it defined in the interface.

Example

```
interface SampleInterface{

  int UPPER_LIMIT = 100;

  //int LOWER_LIMIT; // Error - must be initialised

}

public class InterfaceVariablesExample implements SampleInterface{

  public static void main(String[] args) {

        System.out.println("UPPER LIMIT = " + UPPER_LIMIT);

        // UPPER_LIMIT = 150; // Can not be modified
  }

}
```

Output

```
UPPER_LIMIT = 100;
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as **multiple inheritance.**

Multiple Inheritance in Java

Example

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Output

Hello
Welcome

Interface inheritance

A class implements an interface, but one interface extends another interface.

Example

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output

Hello
Welcome

Que : Difference between abstract class and interface

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}