**1. Can abstract class have constructors in Java?**

Ans: Yes, an abstract class can have a constructor in Java. You can either explicitly provide a constructor to abstract class or if you don't, the compiler will add default constructor of no argument in abstract class. This is true for all classes and it also applies to an abstract class.

**2. Can abstract class implements interface in Java? do they require to implement all methods?**

Ans: In Java, an abstract class can implement an interface, and not provide implementations of all of the interface’s methods. It is the responsibility of the first concrete class that has that abstract class as an ancestor to implement all of the methods in the interface.

**3. Can abstract class be final in Java?**

Ans: No a class cannot be marked "abstract" as well as "final". Marking a class "abstract" means it contains partial implementation of methods and hence depends on its subclasses to provide concrete implementation.

Marking the class "final" means that no other class can extend it and hence abstract classes cannot be marked final.

**4. Can abstract class have static methods in Java?**

Ans: Yes But you need to define the static method. You cannot create an abstract static method.

**5. Can you create instance of abstract class?**

Ans: No

**6. Is it necessary for abstract class to have abstract method?**

Ans: No

**7. Difference between abstract class and interface in Java?**

Ans:

i. Abstract classes can have constants, members, abstract methods (methods without a body) and defined methods, whereas interfaces can only have constants and methods stubs.

ii. Methods and members of an abstract class can be defined with any visibility, whereas all methods of an interface must be defined as public (they are defined public by default).

iii. When inheriting an abstract class, a concrete child class must define the abstract methods, whereas an abstract class can extend another abstract class and abstract methods from the parent class don't have to be defined.

iv. Similarly, an interface extending another interface is not responsible for implementing methods from the parent interface. This is because interfaces cannot define any implementation.

v. A child class can only extend a single class (abstract or concrete), whereas an interface can extend or a class can implement multiple other interfaces.

vi. A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility (public).

**8. When do you favor abstract class over interface?**

Ans:

> Consider using abstract classes if any of these statements apply to your situation:

i. You want to share code among several closely related classes.

ii. You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).

iii. You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

> Consider using interfaces if any of these statements apply to your situation:

i. You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.

ii. You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.

iii. You want to take advantage of multiple inheritance of type.

**9. What is abstract method in Java?**

Ans: An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this: abstract void moveTo(double deltaX, double deltaY);

**10. Can abstract class contains main method in Java ?**

Ans: Yes

**11. what is static block in java?**

Ans: Static block is used for initializing the static variables.This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

**12. What is the need of static block?**

Ans: Static block is used for initializing the static variables.

**13. Can we overload static methods in java?**

Ans: Yes. We can overload static method in java

abstract class Test{

static void add(){System.out.println("in add()");}

static void add(int a){System.out.println("overloaded add(a)");}

}

public class MainClass extends Test {

public static void main(String[] args) throws IOException {

Test.add();

Test.add(10);

}

O/P:

In add()

In overloaded add(a)

**14. Can we call super class static methods from sub class?**

Ans: Yes.

class Test { static void add() { System.out.println("in add()");}}

public class MainTest extends Test{

public static void main(String... a){Test.add();}

O/P:

in add()

**15. What is the difference between final and static keywords?**

Ans:

static: static keyword can be applied to instance variables and methods but not to classes. When applied, variables and methods can be called without the help of an object. When a method or variable is called without object, encapsulation is not maintained. That is, with static variables and methods, encapsulation does not exist.

final: final keyword can be applied to all constructs – variables, methods and classes. When applied, final behaves very differently with each with different functionalities.

final variables: must initialized and cannot be modified. Acts as a constanst

final methods: cannot be override

final class: cannot inherite

**16. Write a note on covariant return type with example code.**

Ans:

Before JDK 5.0, it was not possible to override a method by changing the return type. When we override a parent class method, the name, argument types and return type of the overriding method in child class has to be exactly same as that of parent class method. Overriding method was said to be invariant with respect to return type.

**Covariant return types**

Java 5.0 onwards it is possible to have different return type for overriding method in child class, but child’s return type should be sub-type of parent’s return type. Overriding method becomes variant with respect to return type.

Co-variant return type is based on Liskov substitution principle.

Example:

// Java program to demonstrate that we can have

// different return types if return type in

// overridden method is sub-type

// Two classes used for return types.

class A {}

class B extends A {}

class Base

{

A fun()

{

System.out.println("Base fun()");

return new A();

}

}

class Derived extends Base

{

B fun()

{

System.out.println("Derived fun()");

return new B();

}

}

public class Main

{

public static void main(String args[])

{

Base base = new Base();

base.fun();

Derived derived = new Derived();

derived.fun();

}

}

O/P:

Base fun()

Derived fun()

Advantages:

It helps to avoid confusing type casts present in the class hierarchy and thus making the code readable, usable and maintainable.

We get a liberty to have more specific return types when overriding methods.

Help in preventing run-time ClassCastExceptions on returns

**17. Write a note on Enum with example code.**

Ans: An enum type is a special data type that enables for a variable to be a set of predefined constants. They are constants, the names of an enum type's fields are in uppercase letters.

Ex: public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY

}

public class EnumTest {

Day day;

public EnumTest(Day day) {

this.day = day;

}

public void tellItLikeItIs() {

switch (day) {

case MONDAY:

System.out.println("Mondays are bad.");

break;

case FRIDAY:

System.out.println("Fridays are better.");

break;

case SATURDAY: case SUNDAY:

System.out.println("Weekends are best.");

break;

default:

System.out.println("Midweek days are so-so.");

break;

}

}

public static void main(String[] args) {

EnumTest firstDay = new EnumTest(Day.MONDAY);

firstDay.tellItLikeItIs();

EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);

thirdDay.tellItLikeItIs();

EnumTest fifthDay = new EnumTest(Day.FRIDAY);

fifthDay.tellItLikeItIs();

EnumTest sixthDay = new EnumTest(Day.SATURDAY);

sixthDay.tellItLikeItIs();

EnumTest seventhDay = new EnumTest(Day.SUNDAY);

seventhDay.tellItLikeItIs();

}

}

O/P:

Mondays are bad.

Midweek days are so-so.

Fridays are better.

Weekends are best.

Weekends are best.

**18. Write a note on use of super keyword and super() method.**

Ans: The super keyword in java is a reference variable which is used to refer immediate parent class object.

Ex. super() to call parent class constructor

class Animal{

Animal(){System.out.println("animal is created");}

}

class Dog extends Animal{

Dog(){

super();

System.out.println("dog is created");

}

}

class TestSuper3{

public static void main(String args[]){

Dog d=new Dog();

}}

O/P:

animal is created

dog is created

**19. Write a code to implement abstraction using interface.**

Ans: An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. Also, the variables declared in an interface are public, static & final by default.

Ex.

interface MyInterface

{

public void method1();

public void method2();

}

class Demo implements MyInterface

{

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new Demo();

obj.method1();

}

}

O/P:

implementation of method1

**20. Write a Java program to sort a numeric array and a string array.**

Ans:

import java.util.Arrays;

public class Exercise1 {

public static void main(String[] args){

int[] my\_array1 = {

1789, 2035, 1899, 1456, 2013,

1458, 2458, 1254, 1472, 2365,

1456, 2165, 1457, 2456};

String[] my\_array2 = {

"Java",

"Python",

"PHP",

"C#",

"C Programming",

"C++"

};

System.out.println("Original numeric array : "+Arrays.toString(my\_array1));

Arrays.sort(my\_array1);

System.out.println("Sorted numeric array : "+Arrays.toString(my\_array1));

System.out.println("Original string array : "+Arrays.toString(my\_array2));

Arrays.sort(my\_array2);

System.out.println("Sorted string array : "+Arrays.toString(my\_array2));

}

}

Output:

Original numeric array : [1789, 2035, 1899, 1456, 2013, 1458, 2458, 1254, 1472, 2365, 1456, 2165, 1457, 2456]

Sorted numeric array : [1254, 1456, 1456, 1457, 1458, 1472, 1789, 1899, 2013, 2035, 2165, 2365, 2456, 2458]

Original string array : [Java, Python, PHP, C#, C Programming, C++]

Sorted string array : [C Programming, C#, C++, Java, PHP, Python]

**21. Write a Java program to sum values of an array.**

Ans:

public class Exercise2 {

public static void main(String[] args) {

int my\_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int sum = 0;

for (int i : my\_array)

sum += i;

System.out.println("The sum is " + sum);

}

}

O/P:

The Sum is 55

22. Write a Java program to remove a specific element from an array.

Ans:

import java.util.Arrays;

public class Exercise7 {

public static void main(String[] args) {

int[] my\_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};

System.out.println("Original Array : "+Arrays.toString(my\_array));

// Remove the second element (index->1, value->14) of the array

int removeIndex = 1;

for(int i = removeIndex; i < my\_array.length -1; i++){

my\_array[i] = my\_array[i + 1];

}

// We cannot alter the size of an array , after the removal, the last and second last element in the array will exist twice

System.out.println("After removing the second element: "+Arrays.toString(my\_array));

}

}

O/P:

Original Array : [25, 14, 56, 15, 36, 56, 77, 18, 29, 49]

After removing the second element: [25, 56, 15, 36, 56, 77, 18, 29, 49, 49]

**23. Write a Java program to reverse an array of integer values.**

Ans:

public class ReverseArray {

public static void main(String[] args) {

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

System.out.println("Array before reverse:");

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

System.out.print(numbers[i] + " ");

}

for (int i = 0; i < numbers.length / 2; i++) {

int temp = numbers[i];

numbers[i] = numbers[numbers.length - 1 - i];

numbers[numbers.length - 1 - i] = temp;

}

System.out.println("\nArray after reverse:");

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

System.out.print(numbers[i] + " ");

}

}

}

O/P:

Array before reverse:

1 2 3 4 5 6 7 8 9 10

Array after reverse:

10 9 8 7 6 5 4 3 2 1

**24. Write a Java program to find the duplicate values of an array of integer values.**

Ans:

public class Exercise12 {

public static void main(String[] args)

{

int[] my\_array = {1, 2, 5, 5, 6, 6, 7, 2};

for (int i = 0; i < my\_array.length-1; i++)

{

for (int j = i+1; j < my\_array.length; j++)

{

if ((my\_array[i] == my\_array[j]) && (i != j))

{

System.out.println("Duplicate Element : "+my\_array[j]);

}

}

}

}

}

O/P:

Duplicate Element : 2

Duplicate Element : 5

Duplicate Element : 6