**Java Interview Q&A**

**Q1. Explain OOPs Concept? What is the different OOPs Concept?**

**Ans:-** There are 6 main OOPs concept we have in java. Those are:

1. **Classes**: Class is a user defined data type. It contains data member and member function. Class defines the properties of an object.

Example: Employee is an Object; It has different property we can say like – Employee Id, Employee name, Employee salary, Employee joining date etc.

1. **Objects**: Object is a real-time runtime entity in OOPs. Object contains data and code to manipulate that data. Each object is associated with class data type.
2. **Encapsulation**: Encapsulation is nothing but the wrapping up of data and methods into a single unit. That data is not directly accessible to outside world or classes, but methods can access those data and manipulate the operation. We have to specify access scope of data while creating a class.

Example:

**class** Employee {

**private** **int** EmpId; //only visible to its member

String EmpName="John";

**public** **abstract** **void** employeeDetails(){

System.***out***.println("Employee Id: "+EmpId);

System.***out***.println("Employee Name: "+ EmpName);

}

}

**public** **class** TestPrograms {

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

Employee e1=**new** Employee();

e1.employeeDetails();

}

}

1. **Abstraction**: Abstraction refers to an act of representing an essential feature without including background details. Abstraction class contains data and abstract and non-abstract methods and the definition of abstract methods we must have to write into subclass. For non-abstract method, we have to write the definition of it in abstract class itself.

**Example:**

**abstract** **class** Employee {

**int** EmpId; //Default access scope- subclass can see this fields

String EmpName="John";

**public** **abstract** **void** employeeDetails();

**public** **void** EnterEmpID() { //non-abstract method

EmpId=101;

System.***out***.println(EmpId);

}

}

**class** Organization **extends** Employee{

**public** **void** employeeDetails() {

System.***out***.println("Employee Id: "+EmpId);

System.***out***.println("Employee Name: "+ EmpName);

}

}

**public** **class** TestPrograms {

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

Employee e1=**new** Organization();

e1.EnterEmpID();

e1.employeeDetails();

}

}

1. **Inheritance**: Inheritance is process of deriving a new class from existing class. Parent class holds the common properties of deriving classes.
2. **Polymorphism**: Polymorphism means the ability to take more than one form. A single method name can be used to handle different set of parameters. We have compile time polymorphism and runtime polymorphism.

**Compile time polymorphism**: Method Overloading

**Runtime polymorphism**: Method Overriding

**Q2. What is unchecked cast from object to type? How to resolved “Type safety” warning?**

**Ans:-** An unchecked cast warning in Java occurs when the compiler cannot verify that a cast is safe at compile time. This can happen when you are casting an object to a type that is not a super type or subtype of the object's actual type.

* To address an unchecked cast warning, you can either suppress the warning using the @SuppressWarnings("unchecked") annotation, or you can modify your code to ensure that the cast is safe.
* The ‘unchecked warnings’ is quite popular warning message in Java. However, if you insist this is an invalid warning, and there are no ways to solve it without compromising the existing program functionality. You may just use **@SuppressWarnings(“unchecked”)** to suppress unchecked warnings in Java.

1. **In Class:** If applied to class level, all the methods and members in this class will ignore the unchecked warnings message.

@SuppressWarnings("unchecked")

**public** **class** JSONFileHandling {

1. **In Method:** If applied to method level, only this method will ignore the unchecked warnings message.

@SuppressWarnings("unchecked")

**public** **void** writeToJSONFile() {

1. **In Property:** If applied to property level, only this property will ignore the unchecked warnings message.

@SuppressWarnings("unchecked")

Map<String, Object> map = (Map<String, Object>)address;

**Q3. What are the different types of variable in Java?**

**Ans:-** Java variable is a name given to a memory location. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution. There are 3 types of variable in java:

1. **Local Variable:**
   * A Variable can be declared and used inside a method or a block or constructor is known as local variable.
   * Local variable can be accessible within specified block only. Outside method cannot access that variable.
   * When the variable declared and initialized a value within a block then that variable scope will be within that block only.
2. **Instance Variable:**
   * Instance variable are non-static and it can be declared inside a class and outside of any method.
   * Instance variables are declared in a class. And instance variables are created when an object of a class is created.
3. **Static Variable:**
   * Static Variable is also known as Class variable.
   * The Variables are declared with static keyword within a class outside of any method.
   * Static Variable will create only one copy of variable it will be share among all the objects.

**Example:**

**class** Variable1{

String name; //Instance variable

**public** Variable1() {

name="John";

}

**static** String *lastName*="Smith"; //Static variable

}

**public** **class** TestPrograms {

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

**int** a=10; //Local variable

System.***out***.println("Local var: "+a);

Variable1 inVarObj = **new** Variable1();

System.***out***.println("Instance var: "+inVarObj.name);

System.***out***.println("Static var: "+ Variable1.*lastName*);

}

**Q4. What are the different access modifiers in Java?**

**Ans:-** Access Modifier defines the scope of the variable, method and class within the package and outside the package. To define the scope of variable and method we have different access modifier. Those are:

1. **Default**: No keyword is required to specify default access modifier of a variable or method. The fields are visible or accessible within the current package only
2. **private:** private keyword is required to specify private access modifier of a fields. The fields are visible or accessible within the current class only. No other classes of same package and subclasses also cannot access these fields.
3. **public**: public keyword is required to specify public access modifier of a fields. The fields are visible or accessible everywhere. Anyone can access public data member and methods.
4. **protected**: protected keyword is required to specify protected access modifier of a fields. The fields are visible or accessible to all the classes and subclasses within same package as well as within the subclasses of other package.

**Q5. What is Constructor? And Different types of constructor?**

**Ans:-** Constructor is called when an object is instantiated.

* Constructor is called when object of class is created.
* Constructor have same name as the class itself.
* It does not specify any return type not even void. Because it return the instance of a class.
* Default constructor provides default values of data types.

**Types:**

1. **Default Constructor:** A constructor does not have parameters is known as Default Constructor.
2. **Parameterized Constructor:** A constructor that has parameters is known as Parameterized constructor.
3. **Copy Constructor:** A Constructor copies the data from one object to other object.

**Example**:

**class** ConstructorTuto{

**int** a,b;

ConstructorTuto(){ //Parameterized Constructor

**this**.a=10; **this**.b=20;

System.***out***.println("Default Constructor: "+**this**.a+" "+**this**.b);

}

ConstructorTuto(**int** a, **int** b){ //Parameterized Constructor

**this**.a=a; **this**.b=b;

System.***out***.println("Paramterized Constructor: "+**this**.a+" "+**this**.b);

}

ConstructorTuto(ConstructorTuto copyConstr){ //Copy Constructor

**this**.a=copyConstr.a; **this**.b=copyConstr.b;

System.***out***.println("Copy Constructor: "+**this**.a+" "+**this**.b);

}

**public** **void** print() {

System.***out***.println(a+" "+b);

}

}

**public** **class** TestProgram1 {

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

ConstructorTuto c1 = **new** ConstructorTuto(); //Called Default Constructor;

ConstructorTuto c2=**new** ConstructorTuto(30,40);//Called ParameterizedConstructor

ConstructorTuto c3 = **new** ConstructorTuto(c2); //Called Copy Constructor

c1.print();

c3.print();

}

}

**Q6. What is Constructor Overloading?**

**Ans:-** When we create default constructor, parameterized constructor and copy constructor for a class then it is known as Constructor overloading. For a class we can have different parameterized constructor with different parameters.

**Q7. What is Inheritance and what are the different types of inheritance?**

**Ans:-** Inheritance is the process of creating a new class from existing class. Parent class holds the common properties. By using inheritance we can achieve code reusability, method overriding, abstraction etc.

**Types:**

* 1. **Single Inheritance:** Here we can create one parent and one derived class.
  2. **Multilevel Inheritance:** A derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.
  3. **Hierarchical Inheritance:** One class serves as a superclass (base class) for more than one subclass.
  4. **Multiple** **Inheritance**: One class can have more than one superclass and inherit features from all parent classes. This can be achieving through Interface.
  5. **Hybrid Inheritance:** It is a combination of two or more inheritance type. This can be achieving through Interface.
* **‘extends’** keyword is used for inheriting one class into another.

**Example:**

**class** ConstructorTuto{

**int** a=10,b=20;

}

**class** Sum **extends** ConstructorTuto{

**int** sum = a+b;

**public** **void** print() {

System.***out***.println("Sum: "+sum);

}

}

**public** **class** TestProgram1 {

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

Sum c1 = **new** Sum();

c1.print();

}

}

**Q8. What is the use of ‘*super*’ keyword?**

**Ans:-** The super keyword in Java is a reference variable that is used to refer to parent class objects

Super keyword used in below context:

1. When a derived class and base class has the same data members
2. When parent and sub class having same named method, to avoid ambiguity we can use super keyword.
3. When we want to call parent class constructor then we can use super keyword.

**Example:**

**class** ConstructorTuto{

**int** a=10,b=20;

ConstructorTuto(**int** a){

**this**.a=a;

}

**public** **void** print() {

System.***out***.println("Sum Parent: "+(a+b));

}

}

**class** Sum **extends** ConstructorTuto{

**int** b=40, a=90;

Sum(**int** a){

**super**(a); //super class constructor

}

**public** **void** print() {

**super**.print(); //super class method method

**int** sum = a + **super**.b; //super class variable

System.***out***.println("Sum Child: "+sum);

}

}

**public** **class** TestProgram1 {

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

Sum c1 = **new** Sum(50);

c1.print();

}

}

Output: Sum Parent: 70

Sum Child: 110

**Q9. What is the use of ‘*this*’ keyword?**

**Ans:-**

* ‘this’ is a reference variable that refers to the current object.
* It can be used to access instance variables and methods of the current object.
* this() used to invoke current class constructor.
* ‘this’ used to invoke current class methods and refer to current class variables.

**Example:**

**class** Sum{

**int** a,b;

Sum(){

**this**(10, 20); //constructor

}

Sum(**int** a, **int** b){

**this**.a=a;

**this**.b=b;

**int** sum = a + b; //constructor

System.***out***.println("Sum Child: "+sum);

}

}

**public** **class** TestProgram1 {

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

**new** Sum();

}

}

**Q10. What is the use of ‘*final’* keyword? Can we override final method? Can we extend final class?**

**Ans:-** final keyword used in different context.

**Final Variable -> to create Constant Variable**

**Final Method -> to prevent Method Overriding**

**Final Class -> to prevent inheritance/cannot be extended to other class**

* Final keyword used to indicate that the variable, method, class cannot be modified or changed.
  1. **Final variables:** When a variable is declared as final, its value cannot be changed once it has been initialized. This is useful for declaring constants or other values that should not be modified.
* We must initialize a final variable with some value.
* The blank final variable can be initialized inside every constructor. If not initialized then compiler throws an error.
* Blank static final variable can be initialized in static block or static method.
  1. **Final methods**: When a method is declared as final, it cannot be overridden by a subclass. This is useful for methods that are part of a class’s public API and should not be modified by subclasses.
  2. **Final classes:** When a class is declared as final, it cannot be extended by a subclass. This is useful for classes that are intended to be used as is and should not be modified or extended.

**Example:**

**final** **class** finalClass{ //Final Class

**public** **void** print() {

System.***out***.println("This is a final class");

}

}

**class** sum{

**final** **int** A=10; //Final Variable

**final** **void** print() { //Final Method

System.***out***.println("A: "+A);

}

}

**public** **class** TestProgram1 {

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

sum s1=**new** sum();

s1.print();

}

}

**Q11. What is the use of ‘*static’* keyword?**

**Ans:-** static keyword is used to share the same variable or method among all objects of a given class.

**The *static* keyword is a non-access modifier in Java that is applicable for the following:**

1. **Static Blocks:** Static block get executed only once when the class gets loaded into memory first time. If you need to do the computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded.
2. **Static Variables:** When a variable is declared as a static, that variable will be shared among all the objects of class. All instance of class will share the same static variable. Static block and variable will execute in order they are present in a program.
3. **Static Methods:** static method can access and process only static data. It allows only static variables. Static variable and methods can be accessed by using Class Name only. We cannot access it using object. Static methods won’t allow non static members (If we try to access non static member in static method compiler throws error - Cannot make a static reference to the non-static field).
4. **Nested Static Class:** A class can be made static only if it is a nested class. We cannot declare a top-level class with a static modifier but can declare nested classes as static. Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

**Example:**

**public** **class** TestProgram1 {

**static** **int** *a*=10; //Static Variable

**int** b=20; //Non-static Variable

**static** {

System.***out***.println("Static Block");

}

**public** **static** **void** print() { //static method

System.***out***.println("Static Method: A= "+*a*);

}

**static** **class** MyNestedClass {

**public** **void** disp(){

System.***out***.println("Nested Class: A = "+*a*);

}

}

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

System.***out***.println("Main Method");

TestProgram1.*print*();

TestProgram1.MyNestedClass n1=**new** TestProgram1.MyNestedClass();

n1.disp();

}

}

**Q12. Can we print something on the console without creating main() method?**

**Ans:- Yes we can, by using static block.**

**Q13. When to use static variables and methods?**

**Ans:-** Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

**Q14. Can we override static methods?**

**Ans:-** Yes. We can override static methods.

**Q15. I have a parent class which contains private variable, public variable, static method and final method and if I extended parent class to child class what are the fields will be accessible and which method will be overridden?**

**Ans:-**

* + 1. First thing is that we cannot access any private fields outside parent class only that class methods can access private fields.
    2. Second thing is public variable can be accessed everywhere in class and other classes.
    3. Final method cannot be overridden
    4. Static method can be overridden
    5. And if we want to access static variable in other classes we have make it public and we can access it using class name.

Example:

**class** test{

**private** **int** a=10;

**public** **static** **int** *c*=30;

**public** **int** b=20;

**public** **static** **void** print() {

System.***out***.println("Static method of parent class");

}

**public** **final** **void** print1() {

System.***out***.println("A: "+a);

}

}

**class** Test2 **extends** test{

**public** **void** print3() {

test.*c*=80;

System.***out***.println("Print3 emthod: "+test.*c*);

}

**public** **static** **void** print() {

System.***out***.println("Static method of child class");

}

}

**public** **class** TestProgram1 {

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

test.*c*=50;

System.***out***.println("Main method: "+test.*c*);

Test2 t2=**new** Test2();

t2.print1();

t2.print3();

test.*print*();

}

}

**Q16. What is Abstraction? What is abstract class, abstract method in java?**

**Ans:-** Abstraction refers to the act of representing an essential feature without including background details. Example: Mobile Phone. We are using a mobile phone but we are not aware about the implementation of mobile functionality. We are also not aware of how they are working.

* Abstract keyword is used to define abstract classes, methods and variable.
* Abstract class cannot be instantiated. Because an abstract methods does not have its implementation in abstract class.
* Abstract class can have both abstract and non-abstract methods.
* Abstract class can have constructor which are used to initialize a variable.
* Abstract class can contain instance variable which can be used by abstract class and subclass.
* Abstract class can implement interface

**Q. Which OPP’s concept have you used in your project?**

**Ans:-**