**Java Interview Questions and Answers**

**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? (Warning “Type safety:”)**

**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 supertype 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;