**Q1. Write a simple Banking System program by using OOPs concept where you can get account Holder name balance etc?**

**Answer:**

```
class BankAccount {
    private String accountHolder;
    private double balance;

    public BankAccount(String accountHolder, double initialBalance) {
        this.accountHolder = accountHolder;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited " + amount + " successfully.");
        } else {
            System.out.println("Invalid amount for deposit.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew " + amount + " successfully.");
        } else {
            System.out.println("Insufficient funds or invalid amount for withdrawal.");
        }
    }

    public double getBalance() {
        return balance;
    }

    public String getAccountHolder() {
        return accountHolder;
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("Shivam kumar", 10000.0);

        System.out.println("Account Holder: " + account.getAccountHolder());
        System.out.println("Balance: " + account.getBalance());

        account.deposit(500.0);
        System.out.println("Balance: " + account.getBalance());

        account.withdraw(200.0);
        System.out.println("Balance: " + account.getBalance());

        account.withdraw(2000.0);
        System.out.println("Balance: " + account.getBalance());
    }
}
```

## Q2. Write a Program where you inherit method from parent class and show method Overridden Concept?

**Answer:**

```
class Animal {
    public void sound() {
        System.out.println("The animal makes a sound.");
    }
}

class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("The cat meows.");
    }
}

class Dog extends Animal {
```

```
    @Override
    public void sound() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Cat cat = new Cat();
        Dog dog = new Dog();

        animal.sound();
        cat.sound();
        dog.sound();
    }
}
```

In this Java program, we have a base class Animal with a sound method. The Cat and Dog classes extend the Animal class and override the sound method with their own implementations.

The @Override annotation is used to indicate that the method is intended to override a method in the parent class.

## Q3. Write a program to show run time polymorphism in java?

**Answer:**

```
class Animal {
    public void sound() {
        System.out.println("The animal makes a sound.");
    }
}

class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("The cat meows.");
    }
}
```

```
class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Animal();
        Animal animal2 = new Cat();
        Animal animal3 = new Dog();

        animal1.sound();
        animal2.sound();
        animal3.sound();
    }
}
```

At runtime, when we call the sound method on each of these objects, the JVM determines the actual type of the object and executes the appropriate implementation of the sound method. This is runtime polymorphism, where the method to be executed is resolved dynamically based on the actual object type rather than the reference type.

## Q4. Write a program to show Compile time polymorphism in java?

**Answer:**

```
class MathUtils {
    public int add(int num1, int num2) {
        return num1 + num2;
    }

    public double add(double num1, double num2) {
        return num1 + num2;
    }

    public int add(int num1, int num2, int num3) {
        return num1 + num2 + num3;
    }
```

```
}

public class Main {
    public static void main(String[] args) {
        MathUtils math = new MathUtils();

        int sum1 = math.add(2, 3);
        double sum2 = math.add(2.5, 3.7);
        int sum3 = math.add(2, 3, 4);

        System.out.println("Sum 1: " + sum1);
        System.out.println("Sum 2: " + sum2);
        System.out.println("Sum 3: " + sum3);
    }
}
```

During compilation, the Java compiler determines the appropriate method to be called based on the number and types of arguments provided. This is compile-time polymorphism, where the decision on which method to call is made at compile-time.

## Q5. Achieve loose coupling in java by using OOPs concept?

**Answer:** In object-oriented programming (OOP), loose coupling refers to a design principle that aims to minimize dependencies between classes or modules. This approach promotes flexibility, maintainability, and reusability in software development. There are several ways to achieve loose coupling in Java using OOP concepts. Here are some commonly used techniques:

1. Abstraction: Encapsulate implementation details and expose only necessary interfaces. By defining abstract classes or interfaces, we can provide a contract that decouples the implementation from the usage. Other classes can interact with the abstract interface rather than directly depending on the concrete implementation.

2. Encapsulation: Use access modifiers (such as private, protected, and public) to hide implementation details and expose only necessary methods or properties. This way, classes are not tightly coupled to the internal workings of other classes. By reducing direct access to internal state, we can achieve loose coupling and ensure that changes to one class do not impact others.

3. Dependency Injection (DI): In DI, dependencies are injected into a class from external sources rather than being created within the class itself. By injecting dependencies through interfaces or constructor parameters, we can easily swap implementations without modifying the dependent classes. This approach decouples the class from specific implementations and makes it easier to change or extend functionality.

4. Inversion of Control (IoC): IoC is closely related to DI and promotes loose coupling by shifting the responsibility of managing dependencies from the class itself to an external container or framework. In IoC,

the flow of control is inverted, and dependencies are resolved and provided by the container. This allows for better separation of concerns and reduces coupling between components.

5. Composition over Inheritance: Favor composition (object composition or aggregation) over inheritance to achieve loose coupling. Rather than inheriting behavior from a base class, we can compose classes by combining different objects with specific functionality. This approach promotes flexibility, as classes can be easily composed and extended without tightly coupling them to a specific inheritance hierarchy.

By applying these OOP principles and techniques, we can achieve loose coupling in our Java code, leading to more modular, maintainable, and extensible software systems.

## Q6. What is the benefit of encapsulation in java?

**Answer**: Encapsulation in Java

Encapsulation in Java is a *process of wrapping code and data together into a single unit,* for example, a capsule which is mixed of several medicines.


We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.

It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.

The encapsulate class is easy to test. So, it is better for unit testing.

The standard IDEs are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

## Q7. Is java a t 100% Object oriented Programming language? If no, why?

**Answer:** Pure Object-Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language which supports or have features which treats everything inside program as objects. It doesn't support primitive datatype (like int, char, float, bool, etc.). There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

1. Encapsulation/Data Hiding
2. Inheritance
3. Polymorphism

4. Abstraction
5. All predefined types are objects.
6. All user defined types are objects.
7. All operations performed on objects must be only through methods exposed at the objects.

Example: Smalltalk

**Why Java is not a Pure Object-Oriented Language?**

Java supports property 1, 2, 3, 4 and 6 but fails to support property 5 and 7 given above. Java language is not a Pure Object-Oriented Language as it contain these properties:

- **Primitive Data Type ex. int, long, bool, float, char, etc as Objects:** Smalltalk is a "pure" object-oriented programming language unlike Java and C++ as there is no difference between values which are objects and values which are primitive types. In Smalltalk, primitive values such as integers, booleans and characters are also objects. In Java, we have predefined types as non-objects (primitive types).

```
int a = 5;

System.out.print(a);
```

- **The static keyword:** When we declares a class as static then it can be used without the use of an object in Java. If we are using static function or static variable then we can't call that function or variable by using dot(.) or class object defying object oriented feature.
- **Wrapper Class:** Wrapper class provides the mechanism to convert primitive into object and object into primitive. In Java, you can use Integer, Float etc. instead of int, float etc. We can communicate with objects without calling their methods. ex. using arithmetic operators.

```
String s1 = "ABC" + "A" ;
```

- Even using Wrapper classes does not make Java a pure OOP language, as internally it will use the operations like Unboxing and Autoboxing. So if you create Integer instead of int and do any mathematical operation on it, under the hoods Java is going to use primitive type int only.

Example:

```
public class BoxingExample
{
   public static void main(String[] args)
   {
        Integer i = new Integer(10);
        Integer j = new Integer(20);
        Integer k = new Integer(i.intValue() + j.intValue());
        System.out.println("Output: "+k);
   }
}
```

- **In the above code, there are 2 problems where Java fails to work as pure OOP:**
    1. While creating Integer class you are using primitive type "int" i.e. numbers 10, 20.
    2. While doing addition Java is using primitive type "int".

## Q8. What are the advantages of abstraction in java?

**Answer: Advantages of Abstraction**

1. It reduces the complexity of viewing things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only essential details are provided to the user.
4. It improves the maintainability of the application.
5. It improves the modularity of the application.
6. The enhancement will become very easy because without affecting end-users we can be able to perform any type of changes in our internal system.

7. Improves code reusability and maintainability.
8. Hides implementation details and exposes only relevant information.
9. Provides a clear and simple interface to the user.
10. Increases security by preventing access to internal class details.
11. Supports modularity, as complex systems can be divided into smaller and more manageable parts.
12. Abstraction provides a way to hide the complexity of implementation details from the user, making it easier to understand and use.
13. Abstraction allows for flexibility in the implementation of a program, as changes to the underlying implementation details can be made without affecting the user-facing interface.
14. Abstraction enables modularity and separation of concerns, making code more maintainable and easier to debug.

## Q9. What is an abstraction explained with an Example?

**Answer:** In Java, Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essential units are not displayed to the user.
**Ex:** A car is viewed as a car rather than its individual components.

**What is Abstraction in Java?**

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.
Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

**Real-Life Example:**

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes, etc in the car. This is what abstraction is.

**Java Abstract classes and Java Abstract methods**

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods.
4. A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an abstract keyword.
6. There can be no object of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator.
7. An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

**Algorithm to implement abstraction in Java.**

1. Determine the classes or interfaces that will be part of the abstraction.
2. Create an abstract class or interface that defines the common behaviors and properties of these classes.
3. Define abstract methods within the abstract class or interface that do not have any implementation details.
4. Implement concrete classes that extend the abstract class or implement the interface.
5. Override the abstract methods in the concrete classes to provide their specific implementations.
6. Use the concrete classes to implement the program logic.

## Q10.What is the final class in Java?

**Answer:** A final class in Java cannot be inherited or extended, meaning that no subclass can be created from it. In other words, there is no subclass you can find which will inherit the final class in Java. If a class is

complete in nature, then we can make it a final class which tells us that it can't be an abstract class.
If in case anyone tries to inherit the final class to any subclass, the compiler will throw an error.

**How to Create the Final Class in Java**

These classes are also known as the last class because no other class can inherit them.
In Java, some classes like String, Integer, and other wrapper classes are also proclaimed as the final class.

To create the final class in java simply add the final keyword as a prefix to the class as shown in the syntax of a class given below –

```
final class className

{

  // Body of class

}
```

This is how any final class is created in java.

**When and How to Use the Final Class in Java**

Before using final class, we should have a deep clarity about what is final class in java, we can use final class in mainly two ways –

1. **To prevent inheritance** –a final class cannot be extended or inherited.

```
final class X

{

    // body of class X

}

class Y extends X

{

   // body of class Y



}
```

   This syntax of code will give us a COMPILATION ERROR.

2. **To create immutable classes** – We use the final keyword to create an immutable class. It means an immutable class can always be a final class.