**1. What is Encapsulation?**

Encapsulation is one of the four pillars of Object-Oriented Programming (OOP). It is the practice of hiding the internal state and functionality of an object and only exposing a controlled interface. This is done by declaring variables as private and providing public getter and setter methods to access and modify them.

* Example:

class A {

private String password;

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

Encapsulation helps maintain control over data and prevent unauthorized or invalid access.

**2. Is the below code, correct?**

class A {

private String password;

}

class MainRun {

A a = new A();

print(a.password);

}

No, this code is not correct.

* private means that the variable password is not accessible outside the class A.
* Instead, you should use a getter method to access the private variable:

class A {

private String password;

public String getPassword() {

return password;

}

}

class MainRun {

A a = new A();

System.out.println(a.getPassword());

}

**3. Polymorphism:**

Polymorphism allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.

* Types of Polymorphism:
  1. Method Overriding (Runtime Polymorphism): This occurs when a subclass provides a specific implementation of a method already defined in its parent class. It requires inheritance.
  2. Method Overloading (Compile-time Polymorphism): This happens when multiple methods with the same name exist but have different parameters. It does not require inheritance.

**4. Does Interface support Polymorphism?**

Yes, interfaces support polymorphism. When a class implements an interface, it can provide different implementations for the interface’s methods, and you can use the interface reference to call these methods polymorphically.

**5. Arrange the order of classes:**

The order depends on inheritance relationships:

1. **Base class 1**
2. **Base class 2**
3. **Interface** (if implemented by any classes)
4. **Abstract class** (if involved)
5. **MainClass** (where object creation happens)
6. **Person, Student, InternStudent** (with the hierarchical relationship between them)

**6. Dynamic Binding:**

Dynamic binding refers to the process where the method call is resolved at runtime rather than compile time. In the example of hierarchical inheritance:

class Student {

void show() { System.out.println("I am a student"); }

}

class InternStudent extends Student {

void show() { System.out.println("I am an intern student"); }

}

class RegularStudent extends Student {

void show() { System.out.println("I am a regular student"); }

}

class MainRun {

public static void main(String[] args) {

Student s = new InternStudent(); // Dynamic binding

s.show(); // Outputs: I am an intern student

s = new RegularStudent();

s.show(); // Outputs: I am a regular student

}

}

**7. Explicit Casting vs Implicit Casting:**

* **Explicit Casting** (manual conversion):

double d = 10.5;

int i = (int) d; // Manual conversion from double to int

* **Implicit Casting** (automatic conversion):

int i = 10;

double d = i; // Automatically converts int to double

**8. Which statement to use for conditions:**

To include a student if 1) intern 2) yearofpassout is 2024 3) city is Chennai, use if-else statements:

if (isIntern && yearOfPassout == 2024 && city.equals("Chennai")) {

// Include the student

}

**9. Switch (case) Matching:**

Yes, in java (since version 14), switch statements can match cases like:

* case 'a':
* case 1:
* case true:
* case "Monday":

**10. What is a Subclass, Subtype, and Derived Class?**

* **Subclass**: A class that inherits from another class.
* **Subtype**: A type that is a specialization of another (for example, Integer is a subtype of Number).
* **Derived class**: Another name for a subclass.

**11. What is a Generic Method?**

A generic method is a method that can operate on objects of various types while providing compile-time type safety.

public <T> void printArray(T[] inputArray) {

for (T element : inputArray) {

System.out.println(element);

}

}

**12. What is a Concrete Method?**

A concrete method is a method that has an implementation (i.e., it is not abstract). All non-abstract methods are concrete methods.

**13. What is an Abstract Method?**

An abstract method is a method that is declared without an implementation. Subclasses must provide their own implementation for abstract methods.

abstract class Animal {

abstract void sound(); // Abstract method

}

**14. Object Creation Mechanisms:**

In java, we create objects using the new keyword.

ClassName obj = new ClassName();

**15. C++ - OOPs - Pointer with Class - Encapsulation is Cancelled?**

In C++, if you use a pointer and directly access a class's private data members, you can bypass encapsulation. However, this goes against good practices. You should use getter/setter methods to maintain encapsulation.

**16. String Functions:**

Common string functions include:

* length()
* substring()
* indexOf()
* toLowerCase()
* toUpperCase()
* replace()

**17. Constructor Overloading:**

Constructor overloading means defining multiple constructors in a class with different parameter lists.

class A {

A() { } // No argument constructor

A(int x) { } // Constructor with an int parameter

}