Basic Concepts:

1. What is class and object in Java?

   Ans: A class can be defined as a blueprint that describes the behavior that the object of its type supports.

   An object is an instance of a class.

2. Define a class named Car with attributes make, model, and year.

   Ans:

```
Test.java   Operation.java   Value.java   Logical.java   Compa
1
2 public class Car {
3     String make="Ferrari";
4         int model=2018;
5         int year=2022;
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8
9     }
10
11 }
12
```

3. Create a class named 'Student' with String variable 'name' and integer variable 'roll_no'. Assign the value of roll_no as '2' and that of name as "John" by creating an object of the class Student.

```java

public class Student {
    String name;
    int roll_no;

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Student sd= new Student();
        sd.name="John";
        sd.roll_no= 2;
        System.out.println("Name is : "+sd.name+" and roll no is "+sd.roll_no);
    }

}

```

Problems Javadoc Declaration Console

```
<terminated> Student (2) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 2023, 11:11:22 AM – 11:11:22 AM
Name is : John and roll no is 2
```

Constructors:

4. What is a constructor? What are the types of constructor and their uses in Java?

   Ans: A constructor in Java is a special method that is used to initialize objects.

   The types of constructor are:

a. Default Constructor: A constructor that has no parameters is known as a default constructor. It takes no arguments and initializes the object with default values.
b. Parameterized Constructor: A constructor that has parameters is known as a parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

5. What is constructor overloading?

   Ans: Constructor overloading means having more than one constructor with the same name.

6. Can we use 2 constructors in the same class in Java?

   Ans: Yes, it is possible to have multiple constructors in the same class in Java which is known as constructor overloading. Constructor overloading allows a class to have more than one constructor with a different number or types of parameters.

7. Implement a parameterized constructor for the Car class to initialize its attributes during object creation.

Test.java | Operation.java | Value.java | Logical.java | Compare.java | Concatt.java | concate.java

```java
public class Consstructor {
    String make;
    String model;
    int year;

    // Parameterized constructor
    public Consstructor(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }
    // Getter methods (optional)
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }
    public int getYear() {
        return year;
    }
    public static void main(String[] args) {
        // Creating an instance of the Car class using the parameterized constructor
        Consstructor myCar = new Consstructor("Toyota", "Supra", 2023);
        // Accessing attributes using getter methods (optional)
        System.out.println("Make: " + myCar.getMake());
        System.out.println("Model: " + myCar.getModel());
        System.out.println("Year: " + myCar.getYear());
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Consstructor [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (Dec 13, 2023, 11:58:26 AM – 11:58:26 AM) [pid
Make: Toyota
Model: Supra
Year: 2023
```

Methods:

8. What are methods and how do you declare them in Java?

   Ans: A method is a block of code that performs a specific task or operation. Methods are used to define the behavior of objects, and they encapsulate functionality within a class.

   We can declare methods by

   access_modifier return_type method_name(parameter_list) {

   }

9. List out the different types of access modifiers along with their scope in Java.

   Ans: The four different types od access modifiers are:

   a. Public: The member with public access modifier is accessible from any other class in the same project or external projects.

   b. Private: The member with private access modifier is accessible only within the same class. It is not visible to other classes, even in the same package.
   c. Protected: The member with protected access modifier is accessible within the same package and by subclasses, regardless of the package.

d. Default: If no access modifier is specified (i.e., default), the member is accessible only within the same package.

10. What is the difference between user-defined and library methods in Java?

Ans: A user-defined method is a method defined by the user whereas library methods or built-in methods are the methods created by the developers of Java which are available in the form of packages.

11. In which case do you prefer using static methods over instance methods?

Ans: The choice between static methods and instance methods in Java depends on the nature of the functionality you are implementing and how it relates to the overall design of your class.

12. Add a method to the Car class to display the details of the car.

```java
public class Car {
    // Private attributes
    private String make;
    private String model;
    private int year;

    // Public constructor
    public Car(String make, String model, int yea
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Public getter methods
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    // Public setter methods
    public void setMake(String make) {
        this.make = make;
    }

    public void setModel(String model) {
        this.model = model;
```

```java
    public void setMake(String make) {
        this.make = make;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Public method to display details
    public void displayDetails() {
        System.out.println("Car Details:");
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }

    // Example of using the Car class
    public static void main(String[] args) {
        // Creating an instance of Car
        Car myCar = new Car("Toyota", "Camry", 2022);

        // Using the displayDetails method to show car deta
        myCar.displayDetails();
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> Car [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 202
Car Details:
Make: Toyota
Model: Camry
Year: 2022
```

13. Create multiple Car objects and call the display method for each.

```java
public class Car {
    // Private attributes
    private String make;
    private String model;
    private int year;

    // Public constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Public getter methods
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    // Public setter methods
    public void setMake(String make) {
        this.make = make;
    }

```

```java
    }

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Public method to display details
    public void displayDetails() {
        System.out.println("Car Details:");
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println();
    }

    // Example of using the Car class
    public static void main(String[] args) {
        // Creating multiple instances of Car
        Car car1 = new Car("Toyota", "Camry", 2022);
        Car car2 = new Car("Honda", "Accord", 2021);
        Car car3 = new Car("Ford", "Mustang", 2023);

        // Calling the displayDetails method for each car
        car1.displayDetails();
        car2.displayDetails();
        car3.displayDetails();
    }
}
```

Problems Javadoc Declaration Console

```
terminated> Car [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 2023, 1
Year: 2021

Car Details:
Make: Ford
Model: Mustang
Year: 2023
```

Encapsulation:

14. What is Encapsulation in Java? Why is it called Data hiding?

    Ans: Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.

    Encapsulation helps in hiding the internal implementation details of an object and exposing only what is necessary for the outside world to interact with the object.

15. How to achieve encapsulation in Java? Give an example.

    Ans: Encapsulation is achieved by using access modifiers to control the visibility of class members (fields, methods, and nested classes.

```java
public class Carr {
    // Private attributes
    private String make;
    private String model;
    private int year;

    // Public constructor
    public Carr(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Public getter methods
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    // Public setter methods
    public void setMake(String make) {
        this.make = make;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }
```

```java
    // Example of using the Car class
    public static void main(String[] args) {
        // Creating an instance of Car
        Carr myCarr = new Carr("Toyota", "Camry", 2022);

        // Using getter methods to access attributes
        System.out.println("Make: " + myCarr.getMake());
        System.out.println("Model: " + myCarr.getModel());
        System.out.println("Year: " + myCarr.getYear());

        // Using setter methods to modify attributes
        myCarr.setYear(2023);
        System.out.println("Updated Year: " + myCarr.getYear());
    }
}
```

```
<terminated> Carr [Java Applicati
Make: Toyota
Model: Camry
Year: 2022
Updated Year: 2023
```

16. Create a class User with private attributes like username, password, and email.

```java
public class User {
    // Private attributes
    private String username;
    private String password;
    private String email;

    // Public constructor
    public User(String username, String password, String email) {
        this.username = username;
        this.password = password;
        this.email = email;
    }

    // Public getter methods
    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getEmail() {
        return email;
    }

    // Public setter methods
    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }
```

```java
    public void setPassword(String password) {
        this.password = password;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    // Example of using the User class
    public static void main(String[] args) {
        // Creating an instance of User
        User user1 = new User("john_doe", "securePass123", "john@example.com");

        // Using getter methods to access attributes
        System.out.println("Username: " + user1.getUsername());
        System.out.println("Password: " + user1.getPassword());
        System.out.println("Email: " + user1.getEmail());

        // Using setter methods to modify attributes
        user1.setPassword("newSecurePass456");
        user1.setEmail("john.doe@example.com");
        System.out.println("Updated Password: " + user1.getPassword());
        System.out.println("Updated Email: " + user1.getEmail());
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> User [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 2023, 12:15:42 PM – 12:15:42 PM) [pid:
Username: john_doe
Password: securePass123
Email: john@example.com
Updated Password: newSecurePass456
Updated Email: john.doe@example.com
```

17. Provide public getter and setter methods for each attribute. (refer to qno.16)

```java
public class User {
    // Private attributes
    private String username;
    private String password;
    private String email;

    // Public constructor
    public User(String username, String password, String email) {
        this.username = username;
        this.password = password;
        this.email = email;
    }

    // Public getter methods
    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getEmail() {
        return email;
    }

    // Public setter methods
    public void setUsername(String username) {
        this.username = username;
    }
```

String username

Press 'F2' for focus

Test.java | Operation.java | Value.java | Logical.java | concate.java | Car.java | Student.java

```java
    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    // Example of using the User class
    public static void main(String[] args) {
        // Creating an instance of User
        User user1 = new User("john_doe", "securePass123", "john@example.com");

        // Using getter methods to access attributes
        System.out.println("Username: " + user1.getUsername());
        System.out.println("Password: " + user1.getPassword());
        System.out.println("Email: " + user1.getEmail());

        // Using setter methods to modify attributes
        user1.setPassword("newSecurePass456");
        user1.setEmail("john.doe@example.com");
        System.out.println("Updated Password: " + user1.getPassword());
        System.out.println("Updated Email: " + user1.getEmail());
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> User [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 2023, 12:17:14 PM – 12:17:14 PM) [pid: 16
Username: john_doe
Password: securePass123
Email: john@example.com
Updated Password: newSecurePass456
Updated Email: john.doe@example.com
```

18. Make the attributes of the Car class private and provide public methods to access and modify them.

```java
public class Car {
    // Private attributes
    private String make;
    private String model;
    private int year;

    // Public constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Public getter methods
    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    // Public setter methods
    public void setMake(String make) {
        this.make = make;
    }
```

```java

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Public method to display details
    public void displayDetails() {
        System.out.println("Car Details:");
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }

    // Example of using the Car class
    public static void main(String[] args) {
        // Creating an instance of Car
        Car myCar = new Car("Toyota", "Camry", 2022);

        // Using getter methods to access attributes
        System.out.println("Make: " + myCar.getMake());
        System.out.println("Model: " + myCar.getModel());
        System.out.println("Year: " + myCar.getYear());

        // Using setter methods to modify attributes
        myCar.setYear(2023);
        System.out.println("Updated Year: " + myCar.getYear());

```

Problems Javadoc Declaration Console ×

```
<terminated> Car [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Dec 13, 2023, 12:18:52 PM
Model: Camry
Year: 2022
Updated Year: 2023
Car Details:
Make: Toyota
Model: Camry
Year: 2023
```