1. Abstarction:

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

2. Encapsultion:

```
class Mobile1 {
            private int mprice=100;
            private String mname="iPhone";
```

```
	public int getPrice() {
		return mprice;
	}

	public String getName() {
		return mname;
	}
}
public class EncapsulationExample1 {

	public static void main(String[] args) {
		Mobile1 obj= new Mobile1();
		System.out.println(obj.getName()+ ":" +obj.getPrice());
	}

}
```

3. Inheritance:

```
class Appa{	//parent
	public int add(int n1, int n2) {
		return n1+n2;
	}
	public int sub(int n1, int n2) {
		return n1-n2;
	}
}

class Suhas extends Appa{	//child of Appa parent
	public int mul(int n1, int n2) {
		return n1*n2;
```

```
	}
	public int div(int n1, int n2) {
		return n1/n2;
	}
}

class Preetham extends Suhas{   //child of Suhas parent
	public double power(int n1, int n2) {
		return Math.pow(n1, n2);
	}
}

public class InheritanceExample {
	public static void main(String args[]) {
		Appa obj= new Appa();
		Suhas obj1= new Suhas();
		Preetham obj2= new Preetham();
		System.out.println(obj.add(12,10));
		System.out.println(obj1.mul(12,10));
		System.out.println(obj2.power(12,10));
	}
}
```

4. Polymorphism:

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

class Pig extends Animal {
```

```
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

5. Constructors(Default, Parametrized):

```
class Human {
        private int age;
        private String name;

        public Human() {          //default Constructor
                age=12;
                name="Preethu";
        }
```

```
	public Human(int age, String name) {	//parameterized Constructor
		this.age=age;
		this.name=name;
	}
	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
}

public class ConstructorExample {

	public static void main(String[] args) {
		Human obj= new Human();		//default with no parameters
		Human obj1= new Human(22, "Suhas"); //default with parameters
		System.out.println(obj.getName()+ ":" +obj.getAge());
		System.out.println(obj1.getName()+ ":" +obj1.getAge());
	}
```

```
}
```

6. Method Overloading:

```
class Calculator1 {
	public int add(int n1, int n2) {
		return n1+n2;
	}
	public double add(double n1, double n2) {
		return n1+n2;
	}
}

public class Methodoverloading {
	public static void main(String a[]) {
		Calculator1 cal= new Calculator1();
		int res=cal.add(1, 2);
		System.out.println(res);
		double res1=cal.add(2.0, 4.1);
		System.out.println(res1);
	}
}
```

7. Method Overriding:

```
class Electronics {		//same methodname as child with same parameters
	public int display(int n1, int n2) {
		return n1+n2;
	}
}

class SmartPhone extends Electronics {
	public int display(int n1, int n2) {
```

```
                return n1+n2+1;
        }
}

public class MethodOverrridingExample {
        public static void main(String args[]) {
                SmartPhone obj= new SmartPhone();
                System.out.println(obj.display(10, 2));
        }
}
```

8. Thread using extends:

```
class T1 extends Thread{        //we can also set priority using min_priority,max_priority
        public void run() {     //run method
                for(int i=0;i<=50;i++){
                        System.out.println("Hi");
                        try {
                                Thread.sleep(10);       //sleep method
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}

class T2 extends Thread{
        public void run() {
                for(int i=0;i<=50;i++){
                        System.out.println("Hello");
                        try {
                                Thread.sleep(10);
```

```
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}
public class ThreadExample {

        public static void main(String[] args) {
                T1 obj1= new T1();
                T2 obj2= new T2();

                obj1.start();
                obj2.start();

        }

}
```

9. Threads using runnable:

```
class T111 implements Runnable{          //using lambda expression which supports functional interface concept to create threads

}

class T222 implements Runnable{
        public void run() {
                for(int i=0;i<=5;i++){
                        System.out.println("Hello");
                        try {
                                Thread.sleep(10);
```

```
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
            }
        }
}
public class ThreadExample2 {

        public static void main(String[] args) {
            Runnable obj1= () ->    //lambda expression
                    {
                        for(int i=0;i<=5;i++){
                            System.out.println("Hi");
                            try {
                                Thread.sleep(10);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    };
            Runnable obj2= () ->    //lambda expression
                {
                    for(int i=0;i<=5;i++){
                        System.out.println("Hello");
                        try {
                            Thread.sleep(10);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                };
```

```
                Thread t1= new Thread(obj1);
                Thread t2= new Thread(obj2);

                t1.start();
                t2.start();

        }

}
```

10. Reverse a String without using `reverse()`:

```java
public class ReverseString {
    public static void main(String[] args) {
        String str = "HelloWorld";
        String reversed = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed += str.charAt(i);
        }
        System.out.println("Reversed String: " + reversed);
    }
}
```

11. Check if a Number is Prime:

```java
import java.util.Scanner;

public class PrimeCheck {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        boolean isPrime = true;

        if (num <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i <= Math.sqrt(num); i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime) {
            System.out.println(num + " is a prime number.");
        } else {
            System.out.println(num + " is not a prime number.");
        }
    }
}
```

12. Find Factorial Using Recursion:

```java
import java.util.Scanner;

public class FactorialRecursion {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        System.out.println("Factorial: " + factorial(num));
    }

    public static int factorial(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return n * factorial(n - 1);
    }
}
```

13. Basic Calculator:

```java
import java.util.Scanner;

public class BasicCalculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
        System.out.print("Enter operation (+, -, *, /): ");
        char op = sc.next().charAt(0);

        double result = 0;
        switch (op) {
```

```
            case '+': result = num1 + num2; break;
            case '-': result = num1 - num2; break;
            case '*': result = num1 * num2; break;
            case '/':
                if (num2 != 0) result = num1 / num2;
                else System.out.println("Cannot divide by zero");
                break;
            default: System.out.println("Invalid operator");
        }

        System.out.println("Result: " + result);
    }
}
```

14. Find the Largest Element in an Array:

```java
public class LargestInArray {
    public static void main(String[] args) {
        int[] arr = {12, 3, 45, 67, 23, 89};
        int largest = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > largest) {
                largest = arr[i];
            }
        }

        System.out.println("Largest element: " + largest);
    }
}
```

```

### 15. Bubble Sort:

```java
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 5, 6};
        bubbleSort(arr);

        System.out.print("Sorted Array: ");
        for (int i : arr) {
            System.out.print(i + " ");
        }
    }

    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        boolean swapped;
        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) break;
        }
```
```

```
    }
}
```

16. Check if a String is a Palindrome:

```java
import java.util.Scanner;

public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = sc.nextLine();
        String reversed = new StringBuilder(str).reverse().toString();

        if (str.equals(reversed)) {
            System.out.println(str + " is a palindrome.");
        } else {
            System.out.println(str + " is not a palindrome.");
        }
    }
}
```

17. Find Fibonacci Series up to Given Number:

```java
import java.util.Scanner;

public class FibonacciSeries {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter the number of terms: ");
        int n = sc.nextInt();

        int a = 0, b = 1;
        System.out.print("Fibonacci Series: " + a + " " + b);

        for (int i = 2; i < n; i++) {
            int next = a + b;
            System.out.print(" " + next);
            a = b;
            b = next;
        }
    }
}
```

18. Remove Duplicates from an Array:

```java
import java.util.HashSet;
import java.util.Set;

public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 4, 5, 1};
        Set<Integer> set = new HashSet<>();

        for (int i : arr) {
            set.add(i);
        }

        System.out.println("Array without duplicates: " + set);
```

```
    }
}
```

19. Implement a Simple Linked List:

```java
class LinkedList {
    Node head;

    static class Node {
        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }

    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }
```

```
    public void printList() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.insert(10);
        list.insert(20);
        list.insert(30);
        list.printList();
    }
}
```