**Assignments Array and String and opps concept**

**Reverse a String:**

Java

public class ReverseString {

public static String reverse(String str) {

StringBuilder reversedStr = new StringBuilder();

for (int i = str.length() - 1; i >= 0; i--) {

reversedStr.append(str.charAt(i));

}

return reversedStr.toString();

}

public static void main(String[] args) {

String str = "hello";

String reversedStr = reverse(str);

System.out.println(reversedStr); // Output: olleh

}

}

**Check for Palindrome:**

Java

public class PalindromeCheck {

public static boolean isPalindrome(String str) {

int left = 0, right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

public static void main(String[] args) {

String str = "racecar";

if (isPalindrome(str)) {

System.out.println(str + " is a palindrome.");

} else {

System.out.println(str + " is not a palindrome.");

}

}

}

**Remove Duplicates from a String:**

public class RemoveDuplicates {

public static String removeDuplicates(String str) {

StringBuilder result = new StringBuilder();

for (int i = 0; i < str.length(); i++) {

if (result.indexOf(String.valueOf(str.charAt(i))) == -1) {

result.append(str.charAt(i));

}

}

return result.toString();

}

public static void main(String[] args) {

String str = "hello world";

String result = removeDuplicates(str);

System.out.println(result); // Output: helo wrd

}

}

**Array Questions:**

**Find Maximum and Minimum:**

public class MaxMin {

public static void main(String[] args) {

int[] arr = {10, 25, 5, 30, 15};

int max = arr[0], min = arr[0];

for (int i = 1; i < arr.length; i++) {

max = Math.max(max, arr[i]);

min = Math.min(min, arr[i]);

}

System.out.println("Maximum: " + max);

System.out.println("Minimum: " + min);

}

}

**Reverse an Array:**

public class ReverseArray {

public static void reverse(int[] arr) {

int n = arr.length;

for (int i = 0; i < n / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - i - 1];

arr[n - i - 1] = temp;

}

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

reverse(arr);

System.out.println(Arrays.toString(arr)); // Output: [5, 4, 3, 2, 1]

}

}

**Polymorphism in java**

class Animal {

public void makeSound() {

System.out.println("Generic animal sound");

}

public void eat(String food) {

System.out.println("Eating " + food);

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Woof!");

}

public void wagTail() {

System.out.println("Wagging tail");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

public class PolymorphismExample {

public static void main(String[] args) {

Animal animal = new Animal();

Dog dog = new Dog();

Cat cat = new Cat();

// Method Overriding

animal.makeSound(); // Generic animal sound

dog.makeSound(); // Woof!

cat.makeSound(); // Meow!

// Method Overloading

animal.eat("bones"); // Eating bones

animal.eat("fish"); // Eating fish

}

}

**Encapsulation code in java :**

public class Person {

private String name;

private int age;

// Getter method for name

public String getName() {

return name;

}

// Setter method for name

public void setName(String name) {

this.name = name;

}

// Getter method for age

public int getAge() {

return age;

}

// Setter method for age

public void setAge(int age) {

if (age >= 0) {

this.age = age;

} else {

System.out.println("Age cannot be negative.");

}

}

}

1. Inheritance

// Parent class (SuperClass)

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

// Child class (SubClass) inheriting from Animal

class Dog extends Animal {

public void bark() {

System.out.println("Dog is barking.");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

}

}

**single-level inheritance by using class :**

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

public void bark() {

System.out.println("Dog is barking.");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

}

}

**multi-level Inheritance with class Example:**

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

public void bark() {

System.out.println("Dog is barking.");

}

}

class Puppy extends Dog {

public void play() {

System.out.println("Puppy is playing.");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Puppy puppy = new Puppy();

// Accessing methods from all levels of inheritance

puppy.eat(); // Inherited from Animal

puppy.bark(); // Inherited from Dog

puppy.play(); // Specific to Puppy

}

}

**Hierarchical Inheritance with class Example:**

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal {

public void bark() {

System.out.println("Dog is barking.");

}

}

class Cat extends Animal {

public void meow() {

System.out.println("Cat is meowing.");

}

}

public class HierarchicalInheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

Cat cat = new Cat();

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

cat.eat(); // Inherited from Animal

cat.meow(); // Specific to Cat

}

}

**Single level Inheritance with Interface Example:**

interface Animal {

void eat();

}

class Dog implements Animal {

public void eat() {

System.out.println("Dog is eating.");

}

public void bark() {

System.out.println("Dog is barking.");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

}

}

**Multiple level Inheritance with Interface Example:**

interface Animal {

void eat();

}

interface Mammal extends Animal {

void giveBirth();

}

class Dog extends Mammal {

public void eat() {

System.out.println("Dog is eating.");

}

public void giveBirth() {

System.out.println("Dog is giving birth.");

}

public void bark() {

System.out.println("Dog is barking.");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

dog.giveBirth(); // Inherited from Mammal

dog.bark(); // Specific to Dog

}

}

**Hierarchical Inheritance with Interface Example:**

interface Pet {

void makeSound();

}

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal implements Pet {

public void bark() {

System.out.println("Dog is barking.");

}

@Override

public void makeSound() {

System.out.println("Woof!");

}

}

class Cat extends Animal implements Pet {

public void meow() {

System.out.println("Cat is meowing.");

}

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

Cat cat = new Cat();

dog.eat();

dog.bark();

dog.makeSound();

cat.eat();

cat.meow();

cat.makeSound();

}

}

**Multiple Inheritance with Interface Example:**

interface Flyable {

void fly();

}

interface Swimmable {

void swim();

}

class Duck implements Flyable, Swimmable {

public void fly() {

System.out.println("Duck is flying.");

}

public void swim() {

System.out.println("Duck is swimming.");

}

}

public class MultipleInheritanceExample {

public static void main(String[] args) {

Duck duck = new Duck();

duck.fly();

duck.swim();

}

}

**Exm: Hybrid Inheritance :**

which is combination of inheritance is called hybrid inheritance.

interface Flyable {

void fly();

}

interface Swimmable {

void swim();

}

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal implements Swimmable {

public void bark() {

System.out.println("Dog is barking.");

}

public void swim() {

System.out.println("Dog is swimming.");

}

}

class Cat extends Animal implements Flyable {

public void meow() {

System.out.println("Cat is meowing.");

}

public void fly() {

System.out.println("Cat is flying (Imagine a flying cat!).");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Dog dog = new Dog();

Cat cat = new Cat();

// Single Inheritance and Method Overriding

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

dog.swim(); // Implemented from Swimmable

// Multiple Inheritance (Simulated with Interfaces)

cat.eat(); // Inherited from Animal

cat.meow(); // Specific to Cat

cat.fly(); // Implemented from Flyable

}

}

**Exa :**

interface Flyable {

void fly();

}

interface Swimmable {

void swim();

}

class Animal {

public void eat() {

System.out.println("Animal is eating.");

}

}

class Dog extends Animal implements Swimmable {

public void bark() {

System.out.println("Dog is barking.");

}

public void swim() {

System.out.println("Dog is swimming.");

}

}

class Cat extends Animal implements Flyable {

public void meow() {

System.out.println("Cat is meowing.");

}

public void fly() {

System.out.println("Cat is flying (Imagine a flying cat!).");

}

}

public class InheritanceExample {

public static void main(String[] args) {

// Single Inheritance

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

dog.bark(); // Specific to Dog

dog.swim(); // Implemented from Swimmable

// Multiple Inheritance (Simulated with Interfaces)

Cat cat = new Cat();

cat.eat(); // Inherited from Animal

cat.meow(); // Specific to Cat

cat.fly(); // Implemented from Flyable

}

}

**Method Overloading and Overriding:**

public class OverloadingExample {

public void add(int a, int b) {

System.out.println("Sum of integers: " + (a + b));

}

public void add(double a, double b) {

System.out.println("Sum of doubles: " + (a + b));

}

public void add(int a, int b, int c) {

System.out.println("Sum of three integers: " + (a + b + c));

}

public static void main(String[] args) {

OverloadingExample obj = new OverloadingExample();

obj.add(10, 20);

obj.add(10.5, 20.5);

obj.add(10, 20, 30);

}

}

**Method Overriding:**

class Animal {

public void makeSound() {

System.out.println("Generic animal sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Woof!");

}

}

public class OverridingExample {

public static void main(String[] args) {

Animal animal = new Dog();

animal.makeSound(); // Output: Woof!

}

}