Java specific assignments

-------------------------

1. How do you swap two numbers without using a third variable in Java?

package assignments;

public class SwapNumbers {

public static void main(String[] args) {

int a = 5;

int b = 3;

System.out.println("Before swap: a = " + a + ", b = " + b);

a = a + b;

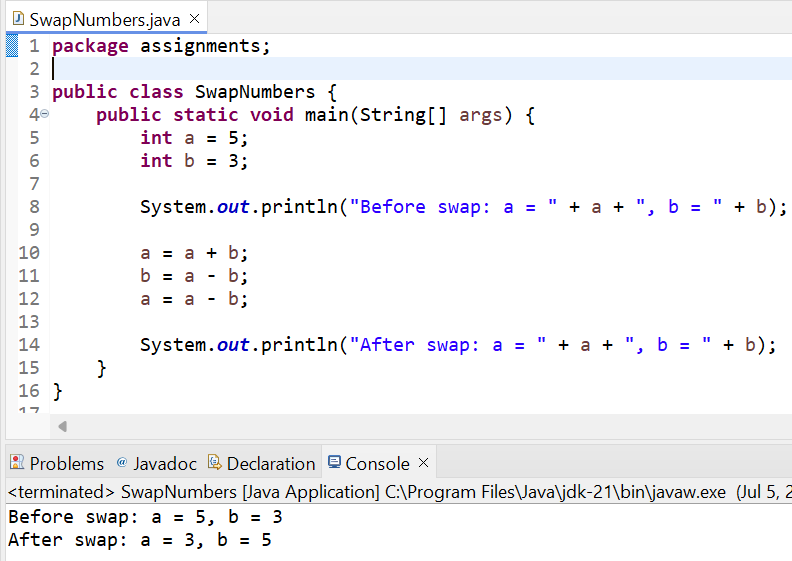
b = a - b;

a = a - b;

System.out.println("After swap: a = " + a + ", b = " + b);

}

}



2. How do you remove spaces from a string in Java

package assignments;

public class RemoveSpaces {

public static void main(String[] args) {

String str = "Hello World, How Are You?";

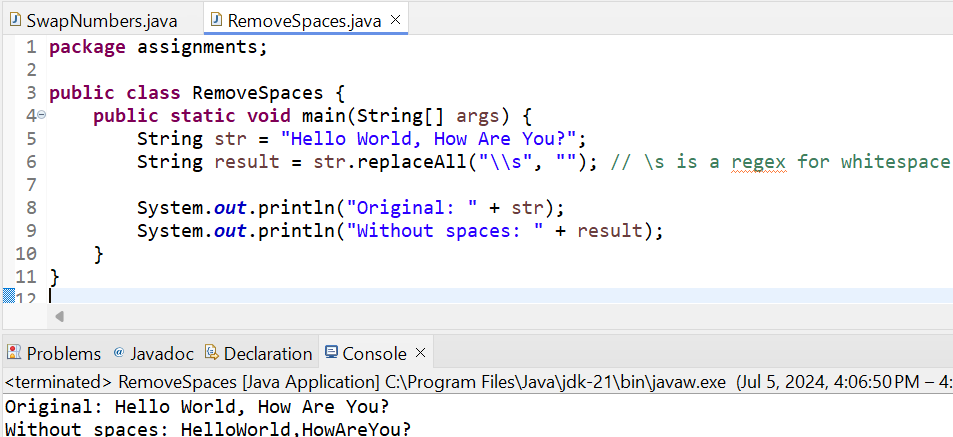
String result = str.replaceAll("\\s", ""); // \s is a regex for whitespace

System.out.println("Original: " + str);

System.out.println("Without spaces: " + result);

}

}



3. Write a Java program to check if a vowel is present in a string.

package assignments;

import java.util.Scanner;

public class VowelChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = scanner.nextLine();

boolean vowelPresent = false;

for (char ch : input.toLowerCase().toCharArray()) {

switch (ch) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

vowelPresent = true;

break;

}

}

if (vowelPresent) {

System.out.println("The string contains a vowel.");

} else {

System.out.println("The string does not contain a vowel.");

}

}

}



4. How do you reverse a linked list in Java?

package assignments;

class LinkedList {

static Node head;

static class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

Node reverse(Node node) {

Node prev = null;

Node current = node;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

return prev;

}

void printList(Node node) {

while (node != null) {

System.out.print(node.data + " ");

node = node.next;

}

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.head = new Node(1);

list.head.next = new Node(2);

list.head.next.next = new Node(3);

list.head.next.next.next = new Node(4);

System.out.println("Original Linked list:");

list.printList(head);

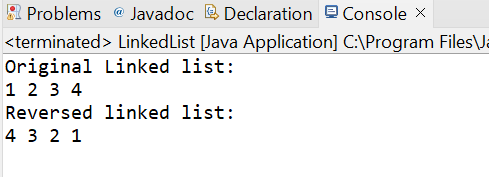
head = list.reverse(head);

System.out.println("\nReversed linked list:");

list.printList(head);

}

}



5. Can you create a pyramid of characters in Java?

package assignments;

public class CharacterPyramid {

public static void main(String[] args) {

int height = 5;

char startingChar = 'A';

printCharacterPyramid(height, startingChar);

}

public static void printCharacterPyramid(int height, char startingChar) {

for (int i = 0; i < height; i++) {

for (int j = 0; j < height - i - 1; j++) {

System.out.print(" ");

}

char currentChar = startingChar;

for (int j = 0; j <= i; j++) {

System.out.print(currentChar);

currentChar++;

}

currentChar -= 2;

for (int j = 0; j < i; j++) {

System.out.print(currentChar);

currentChar--;

}

System.out.println();

}

}

}



6. Write Java program that checks if two arrays contain the same elements.

package assignments;

import java.util.Arrays;

public class ArrayEqualityChecker {

public static void main(String[] args) {

int[] array1 = {1, 2, 3, 4, 5};

int[] array2 = {5, 4, 3, 2, 1};

if (areArraysEqual(array1, array2)) {

System.out.println("The arrays contain the same elements.");

} else {

System.out.println("The arrays do not contain the same elements.");

}

}

public static boolean areArraysEqual(int[] array1, int[] array2) {

if (array1.length != array2.length) {

return false;

}

Arrays.sort(array1);

Arrays.sort(array2);

for (int i = 0; i < array1.length; i++) {

if (array1[i] != array2[i]) {

return false;

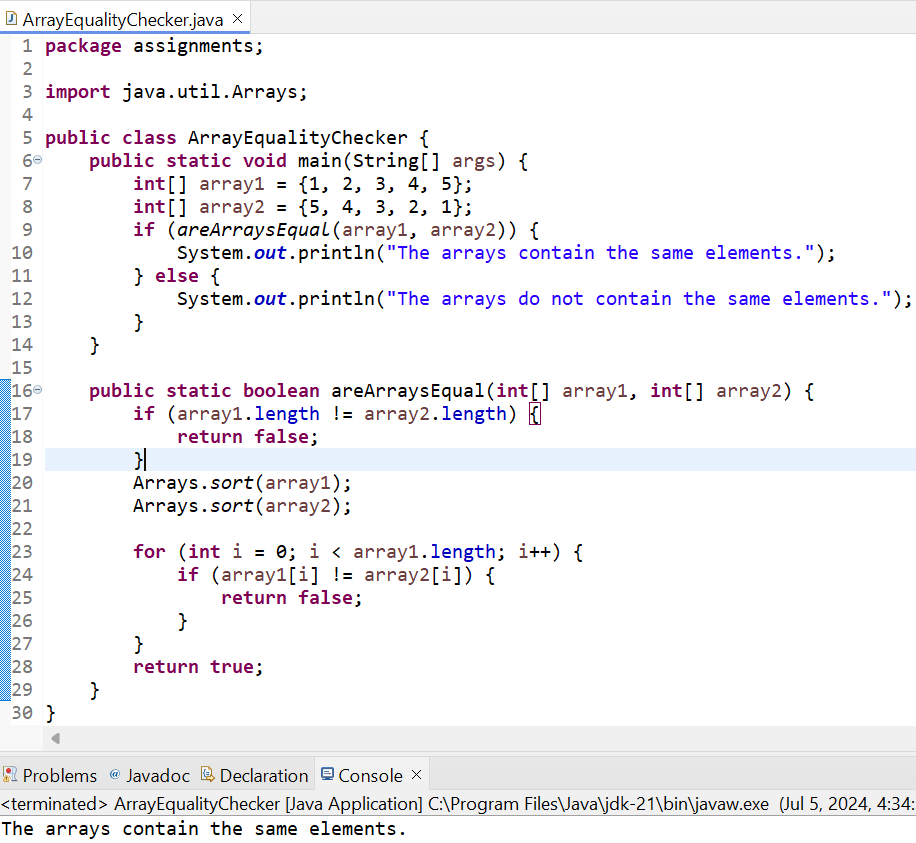
}

}

return true;

}

}



7. How do you get distinct characters and their count in a string in Java?

package assignments;

import java.util.HashMap;

import java.util.Map;

public class CharacterCount {

public static void main(String[] args) {

String str = "Hello, World!";

Map<Character, Integer> charCount = countCharacters(str);

for (Map.Entry<Character, Integer> entry : charCount.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

}

public static Map<Character, Integer> countCharacters(String str) {

Map<Character, Integer> charCount = new HashMap<>();

for (char ch : str.toCharArray()) {

if (charCount.containsKey(ch)) {

charCount.put(ch, charCount.get(ch) + 1);

} else {

charCount.put(ch, 1);

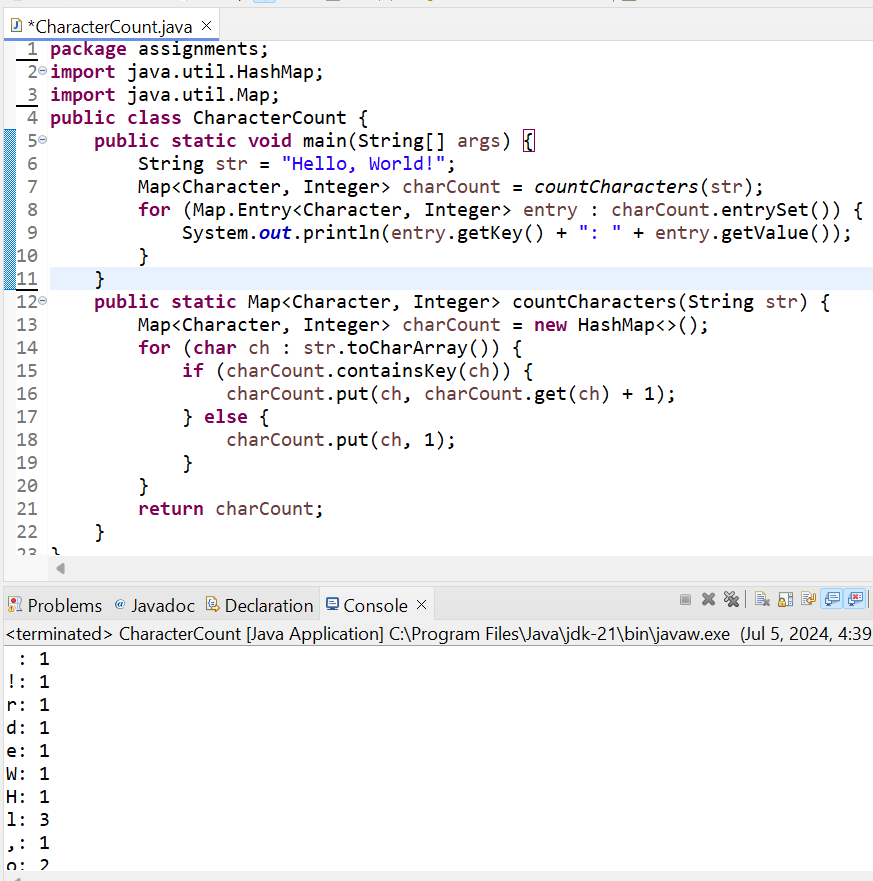
}

}

return charCount;

}

}



8. Can you prove that a String object in Java is immutable programmatically?

package assignments;

public class StringImmutability {

public static void main(String[] args) {

String originalString = "Hello";

System.out.println("Original String: " + originalString);

System.out.println("Original Hash Code: " + originalString.hashCode());

String modifiedString = originalString.concat(" World");

System.out.println("Modified String: " + modifiedString);

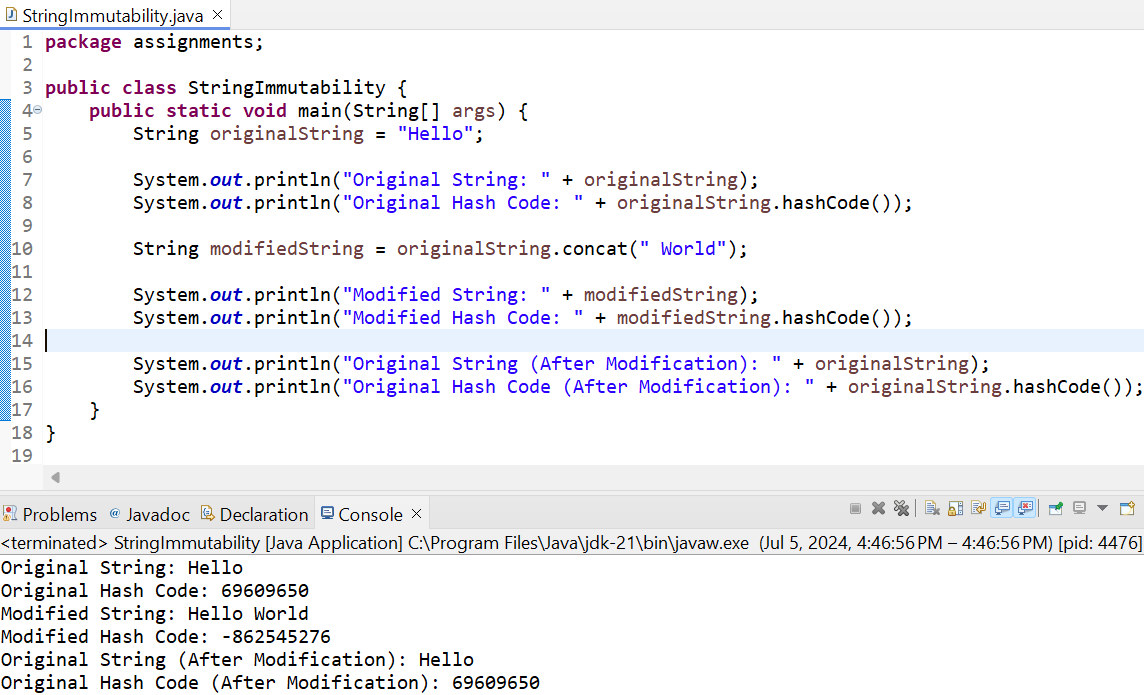
System.out.println("Modified Hash Code: " + modifiedString.hashCode());

System.out.println("Original String (After Modification): " + originalString);

System.out.println("Original Hash Code (After Modification): " + originalString.hashCode());

}

}



* The original string remains "Hello" before and after the modification attempt, and its hash code does not change.
* The modified string "Hello World" is a new String object with a different hash code.

This output clearly demonstrates that the original String object remains unchanged, proving that String objects in Java are immutable.

9. Establish the JDBC connection and perform some SQL statements on MYSQL DataBase

10. Show examples on Overloading & Overriding, Inheritance, Abstraction and Interfaces etc

Overloading

Method overloading allows a class to have more than one method with the same name, but with different parameters.

package assignments;

class Overloading {

public int add(int a, int b) {

return a + b;

}

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

return a + b + c;

}

public double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Overloading obj = new Overloading();

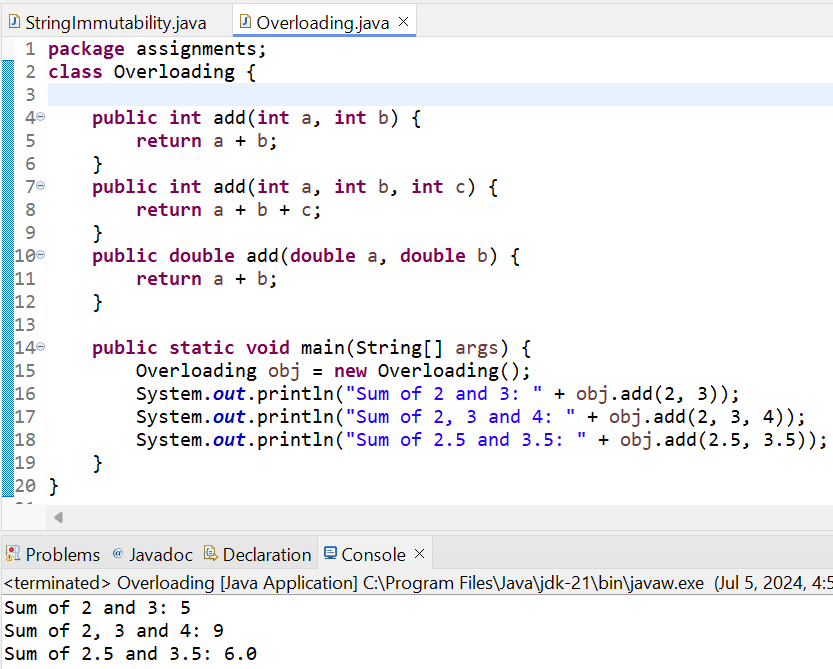
System.out.println("Sum of 2 and 3: " + obj.add(2, 3));

System.out.println("Sum of 2, 3 and 4: " + obj.add(2, 3, 4));

System.out.println("Sum of 2.5 and 3.5: " + obj.add(2.5, 3.5));

}

}



### Overriding

Method overriding allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

package assignments;

class Animal {

public void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Bark");

}

}

public class Overriding {

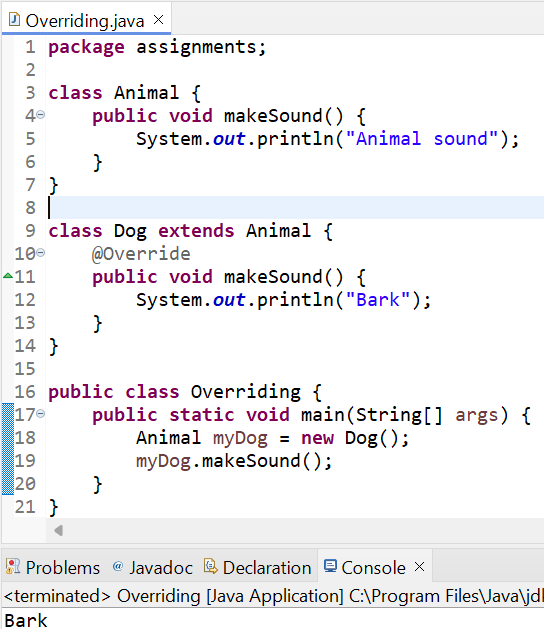
public static void main(String[] args) {

Animal myDog = new Dog();

myDog.makeSound();

}

}



### Inheritance

Inheritance allows a new class to inherit the properties and methods of an existing class.

package assignments;

class Person {

String name;

int age;

public void display() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

}

class Employee extends Person {

double salary;

public void display() {

super.display(); // Calls the display method of the Person class

System.out.println("Salary: " + salary);

}

}

public class Inheritance {

public static void main(String[] args) {

Employee emp = new Employee();

emp.name = "John";

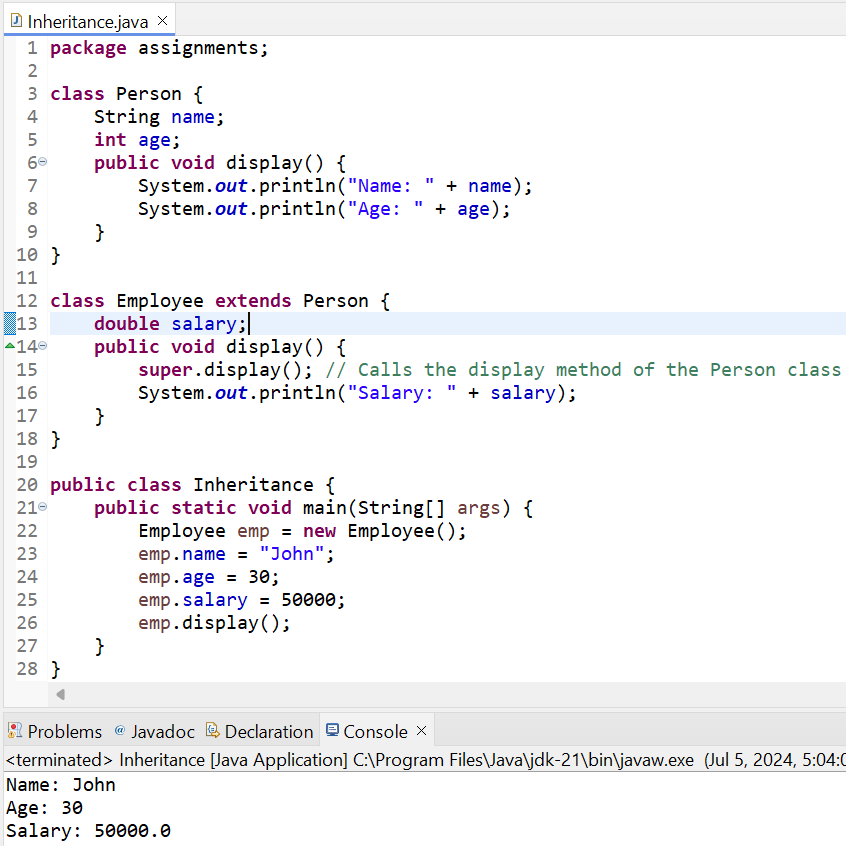
emp.age = 30;

emp.salary = 50000;

emp.display();

}

}



### Abstraction

Abstraction is achieved using abstract classes and interfaces. Abstract classes can have both abstract and non-abstract methods.

package assignments;

//Abstract class

abstract class Shape {

// Abstract method (does not have a body)

public abstract void draw();

// Regular method

public void display() {

System.out.println("Displaying shape");

}

}

//Concrete subclass Circle extending Shape

class Circle extends Shape {

@Override

public void draw() {

System.out.println("Drawing Circle");

}

}

//Concrete subclass Rectangle extending Shape

class Rectangle extends Shape {

@Override

public void draw() {

System.out.println("Drawing Rectangle");

}

}

public class Abstraction {

public static void main(String[] args) {

Shape shape1 = new Circle();

Shape shape2 = new Rectangle();

shape1.draw();

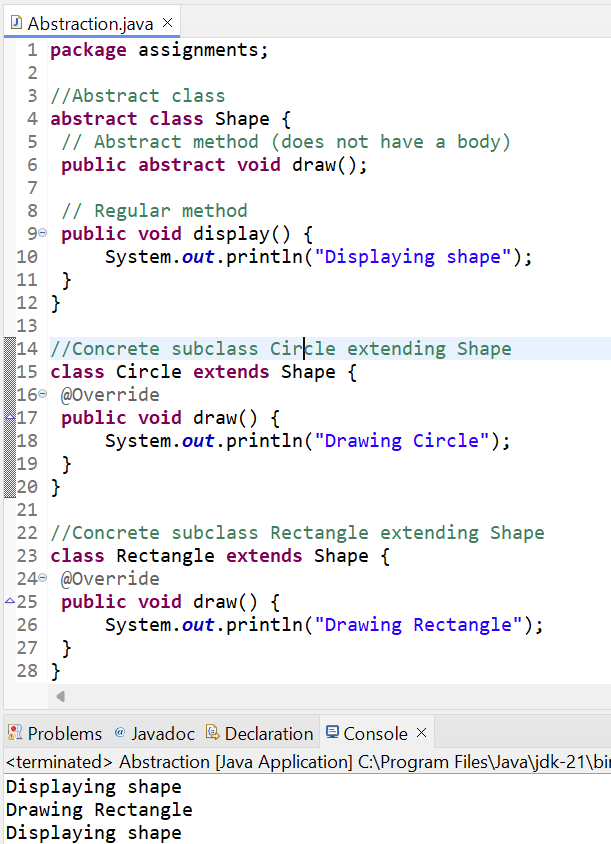
shape1.display();

shape2.draw();

shape2.display();

}

}



### Interfaces

Interfaces are used to achieve abstraction and multiple inheritance in Java.

package assignments;

//Interface for Vehicle

interface Vehicle {

void start();

void stop();

void accelerate();

}

//Concrete class for Car implementing Vehicle

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car started");

}

@Override

public void stop() {

System.out.println("Car stopped");

}

@Override

public void accelerate() {

System.out.println("Car accelerating");

}

}

//Concrete class for Bicycle implementing Vehicle

class Bicycle implements Vehicle {

@Override

public void start() {

System.out.println("Bicycle started");

}

@Override

public void stop() {

System.out.println("Bicycle stopped");

}

@Override

public void accelerate() {

System.out.println("Bicycle speeding up");

}

}

public class Interface {

public static void main(String[] args) {

// Using polymorphism to treat Car and Bicycle as Vehicle

Vehicle myCar = new Car();

Vehicle myBicycle = new Bicycle();

// Calling methods defined in the Vehicle interface

myCar.start();

myCar.accelerate();

myCar.stop();

myBicycle.start();

myBicycle.accelerate();

myBicycle.stop();

}

}

