# Class Practice Problem (Any Four) 1;4;5;6 done

# Practice Problem 1:

Default and Parameterized Constructors

Task: Create a class representing a Book with attributes: title, author, and price.

* Implement a default constructor that sets generic values.
* Implement a parameterized constructor to set all ﬁelds.
* In main, create one Book object using each constructor and display their values.

public class Book { String title;

String author; double price;

// TODO: Default constructor

// TODO: Parameterized constructor

// TODO: Display method

public static void main(String[] args) {

// TODO: Create book1 using default constructor

// TODO: Create book2 using parameterized constructor

// TODO: Display both books

}

}

# public class Book {

# String title;

# String author;

# double price;

# Book() {

# title = "Unknown";

# author = "Unknown";

# price = 0.0;

# }

# Book(String title, String author, double price) {

# this.title = title;

# this.author = author;

# this.price = price;

# }

# void display() {

# System.out.println("Title: " + title + ", Author: " + author + ", Price: " + price);

# }

# public static void main(String[] args) {

# Book book1 = new Book();

# Book book2 = new Book("Java Basics", "John Doe", 299.99);

# book1.display();

# book2.display();

# }

# }

# 

# 🛠 PRACTICE PROBLEM 2: Gaming Controller Conﬁguration System

Default and Parameterized Constructors

Task: Create a gaming controller conﬁguration system that demonstrates constructor patterns without typical examples.

public class GameController {

// TODO: Instance variables for controller configuration private String controllerBrand;

private String connectionType; private boolean hasVibration; private int batteryLevel;

private double sensitivity;

// TODO: Default constructor - creates standard gaming setup public GameController() {

// TODO: Set default values:

// - brand: "GenericPad"

// - connection: "USB"

// - vibration: true

// - battery: 100

// - sensitivity: 1.0

}

// TODO: Parameterized constructor for custom configuration public GameController(String controllerBrand, String

connectionType,

boolean hasVibration, int batteryLevel,

double sensitivity) {

// TODO: Initialize all fields with provided values

// TODO: Validate battery level (0-100) and sensitivity (0.1-3.0)

}

// TODO: Two-parameter convenience constructor



public GameController(String brand, String connectionType) {

// TODO: Set provided values and use defaults for others

}

// TODO: Methods to test functionality public void calibrateController() {

System.out.println("Calibrating " + controllerBrand + " controller...");

}

public void displayConfiguration() {

// TODO: Print all controller settings

}

public void testVibration() { if (hasVibration) {

System.out.println("\*BUZZ\* Vibration test successful!");

} else {

System.out.println("Vibration disabled on this controller.");

}

}

public static void main(String[] args) {

// TODO: Create controller with default constructor

// TODO: Create controller with full parameterized constructor

// TODO: Create controller with convenience constructor

// TODO: Test all methods on each controller

// TODO: Compare different configurations

System.out.println("=== GAMING CONTROLLER SETUP ===");

// Your implementation here

}

}



# 🛠 PRACTICE PROBLEM 3: Music Production Studio Equipment

Constructor Overloading and this() Chaining

Task: Build a music equipment management system demonstrating constructor chaining patterns.

public class AudioMixer {

private String mixerModel;

private int numberOfChannels;

private boolean hasBluetoothConnectivity; private double maxVolumeDecibels;

private String[] connectedDevices; private int deviceCount;

// TODO: No-argument constructor using this() chaining public AudioMixer() {

// TODO: Call three-parameter constructor with defaults:

// - model: "StandardMix-8"

// - channels: 8

// - bluetooth: false

}

// TODO: Two-parameter constructor using this() chaining

public AudioMixer(String mixerModel, int numberOfChannels) {

// TODO: Call three-parameter constructor with bluetooth disabled

}

// TODO: Three-parameter constructor using this() chaining public AudioMixer(String mixerModel, int numberOfChannels,

boolean hasBluetoothConnectivity) {

// TODO: Call main constructor with default max volume

(120.0)



}

// TODO: Main constructor - all parameters

public AudioMixer(String mixerModel, int numberOfChannels, boolean hasBluetoothConnectivity, double

maxVolumeDecibels) {

// TODO: Initialize all fields

// TODO: Initialize connectedDevices array based on numberOfChannels

// TODO: Set deviceCount to 0

// TODO: Print constructor execution message

}

public void connectDevice(String deviceName) {

if (deviceCount < connectedDevices.length) {

connectedDevices[deviceCount] = deviceName; deviceCount++;

System.out.println("Connected: " + deviceName);

} else {

System.out.println("All channels occupied!");

}

}

public void displayMixerStatus() {

System.out.println("\n=== " + mixerModel + " STATUS ==="); System.out.println("Channels: " + numberOfChannels);

System.out.println("Bluetooth: " + (hasBluetoothConnectivity

? "Enabled" : "Disabled"));

System.out.println("Max Volume: " + maxVolumeDecibels + "

dB");

System.out.println("Connected Devices: " + deviceCount + "/"

+ numberOfChannels);

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

System.out.println(" Channel " + (i + 1) + ": " + connectedDevices[i]);

}



}

public static void main(String[] args) {

System.out.println("=== MUSIC STUDIO SETUP ===");

// TODO: Create mixer using no-argument constructor

// TODO: Create mixer using two-parameter constructor

// TODO: Create mixer using three-parameter constructor

// TODO: Create mixer using full constructor

// TODO: Connect different devices to each mixer

// TODO: Display status of all mixers

// TODO: Comment on constructor chaining execution order

}

}

# Practice Problem 4:Smart Home Device Network

this Keyword Usage and Constructor Parameter Disambiguation

Task: Create a smart device network system showcasing extensive this keyword usage.

public class SmartDevice {

private String deviceName;

private String location; private boolean isOnline;

private double powerConsumption;

private String[] connectedDevices; private int connectionCount;

// TODO: Constructor with parameter names matching field names public SmartDevice(String deviceName, String location,

boolean isOnline, double powerConsumption) {

// TODO: Use this keyword to distinguish between parameters and fields

// TODO: Initialize connectedDevices array (size 5)

// TODO: Set connectionCount to 0

}

// TODO: Method using this for parameter disambiguation public void updateLocation(String location) {

// TODO: Use this.location to assign parameter value System.out.println(this.deviceName + " moved to " +

this.location);

}

public void updatePowerConsumption(double powerConsumption) {

// TODO: Use this keyword when parameter name matches field System.out.println("Power consumption updated for " +

this.deviceName);

}

// TODO: Method returning this for chaining

public SmartDevice setOnline(boolean isOnline) {

// TODO: Use this keyword and return this for method chaining this.isOnline = isOnline;

return this;

}

public SmartDevice connectToDevice(String deviceName) {

// TODO: Add device to connectedDevices array

if (this.connectionCount < this.connectedDevices.length) {

this.connectedDevices[this.connectionCount] = deviceName; this.connectionCount++;

System.out.println(this.deviceName + " connected to " + deviceName);

}

return this; // Enable method chaining

}

public SmartDevice rename(String deviceName) {

// TODO: Use this keyword for disambiguation String oldName = this.deviceName;

this.deviceName = deviceName;

System.out.println("Device renamed from " + oldName + " to "

+ this.deviceName);

return this;

}

public void displayDeviceInfo() {

// TODO: Use this keyword to access instance variables

System.out.println("\n=== " + this.deviceName + " INFO ==="); System.out.println("Location: " + this.location);

System.out.println("Status: " + (this.isOnline ? "Online" : "Offline"));

System.out.println("Power: " + this.powerConsumption + "W"); System.out.println("Connections: " + this.connectionCount);

for (int i = 0; i < this.connectionCount; i++) {

System.out.println(" -> " + this.connectedDevices[i]);

}

}

// TODO: Method that calls other methods using this public void performInitialSetup() {

// TODO: Use this to call other methods this.setOnline(true);

System.out.println(this.deviceName + " initial setup

completed");

}

public static void main(String[] args) {

System.out.println("=== SMART HOME DEVICE NETWORK ===");

// TODO: Create devices with parameter names matching field

names

// TODO: Test method chaining using returned this

// TODO: Demonstrate this keyword in various contexts

// TODO: Show parameter disambiguation scenarios

// Example of method chaining:

//

device.setOnline(true).connectToDevice("Alexa").rename("Kitchen Hub");

}

}

public class SmartDevice {

private String deviceName;

private String location;

private boolean isOnline;

private double powerConsumption;

private String[] connectedDevices;

private int connectionCount;

public SmartDevice(String deviceName, String location, boolean isOnline, double powerConsumption) {

this.deviceName = deviceName;

this.location = location;

this.isOnline = isOnline;

this.powerConsumption = powerConsumption;

this.connectedDevices = new String[5];

this.connectionCount = 0;

}

public void updateLocation(String location) {

this.location = location;

System.out.println(this.deviceName + " moved to " + this.location);

}

public void updatePowerConsumption(double powerConsumption) {

this.powerConsumption = powerConsumption;

System.out.println("Power consumption updated for " + this.deviceName);

}

public SmartDevice setOnline(boolean isOnline) {

this.isOnline = isOnline;

return this;

}

public SmartDevice connectToDevice(String deviceName) {

if (this.connectionCount < this.connectedDevices.length) {

this.connectedDevices[this.connectionCount++] = deviceName;

System.out.println(this.deviceName + " connected to " + deviceName);

}

return this;

}

public SmartDevice rename(String deviceName) {

String oldName = this.deviceName;

this.deviceName = deviceName;

System.out.println("Device renamed from " + oldName + " to " + this.deviceName);

return this;

}

public void displayDeviceInfo() {

System.out.println("\n=== " + this.deviceName + " INFO ===");

System.out.println("Location: " + this.location);

System.out.println("Status: " + (this.isOnline ? "Online" : "Offline"));

System.out.println("Power: " + this.powerConsumption + "W");

System.out.println("Connections: " + this.connectionCount);

for (int i = 0; i < this.connectionCount; i++) {

System.out.println(" -> " + this.connectedDevices[i]);

}

}

public void performInitialSetup() {

this.setOnline(true);

System.out.println(this.deviceName + " initial setup completed");

}

public static void main(String[] args) {

SmartDevice device = new SmartDevice("Hub", "Living Room", false, 15.5);

device.performInitialSetup();

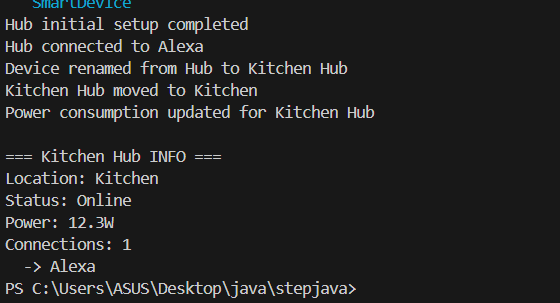
device.connectToDevice("Alexa").rename("Kitchen Hub").updateLocation("Kitchen");

device.updatePowerConsumption(12.3);

device.displayDeviceInfo();

}

}



# Practice Problem 5:

static Keyword Usage

Task: Implement a class Counter that tracks how many objects have been created.

* Use a static variable for the object count.
* Write a static method to return the count.
* In main, create several objects and show the count.

public class Counter {

static int count = 0;

Counter() {

// TODO: Increment count

}

// TODO: Static method getCount()

public static void main(String[] args) {

// TODO: Create several Counter objects

// TODO: Display number of objects created

}

}

public class Counter {

static int count = 0;

Counter() {

count++;

}

static int getCount() {

return count;

}

public static void main(String[] args) {

Counter c1 = new Counter();

Counter c2 = new Counter();

Counter c3 = new Counter();

System.out.println("Number of Counter objects: " + Counter.getCount());

}

}

A screen shot of a computer

AI-generated content may be incorrect.

# Practice Problem 6:

instanceof for Type Checking

Task: Create classes Animal, Dog, and Cat (both extend Animal).

* In main, create an array of Animal containing Dog and Cat objects.
* Use instanceof to count how many Dog and Cat objects in the array.

class Animal {}

class Dog extends Animal {} class Cat extends Animal {}

public class TestInstanceof {

public static void main(String[] args) {

Animal[] animals = { new Dog(), new Cat(), new Dog(), new Animal() };

// TODO: Count number of Dog and Cat instances using instanceof

// TODO: Print results

}

}

class Animal {}

class Dog extends Animal {}

class Cat extends Animal {}

public class TestInstanceof {

public static void main(String[] args) {

Animal[] animals = { new Dog(), new Cat(), new Dog(), new Animal() };

int dogCount = 0;

int catCount = 0;

for (Animal a : animals) {

if (a instanceof Dog) {

dogCount++;

} else if (a instanceof Cat) {

catCount++;

}

}

System.out.println("Number of Dogs: " + dogCount);

System.out.println("Number of Cats: " + catCount);

}

}

