Wrapper classes

1. Check if character is a Digit
2. Compare two Strings
3. Convert using valueof method
4. Create Boolean Wrapper usage
5. Convert null to wrapper classes

Pass by value and pass by reference

Program:

public class WrapperClassDemo {

public static void main(String[] args) {

// 1. Check if character is a Digit

char ch = '5';

System.out.println("Is '" + ch + "' a digit? " + Character.isDigit(ch));

// 2. Compare two Strings

String str1 = "Hello";

String str2 = "World";

System.out.println("Comparing \"" + str1 + "\" and \"" + str2 + "\": " + str1.compareTo(str2));

// 3. Convert using valueOf method

int num = 100;

String numStr = String.valueOf(num);

System.out.println("Converted int to String using valueOf: " + numStr);

// 4. Create Boolean Wrapper usage

Boolean boolObj = Boolean.valueOf("true");

System.out.println("Boolean value from String: " + boolObj);

System.out.println("Boolean true as primitive: " + boolObj.booleanValue());

// 5. Convert null to wrapper classes

try {

String nullString = null;

Integer nullInt = Integer.valueOf(nullString); // This will throw NumberFormatException

System.out.println(nullInt);

} catch (NumberFormatException e) {

System.out.println("Cannot convert null to Integer wrapper: " + e);

}

// Boolean wrapper with null

Boolean nullBool = Boolean.valueOf(null); // returns false, no exception

System.out.println("Boolean from null String: " + nullBool);

}

}

Output:

Is '5' a digit? true

Comparing "Hello" and "World": -15

Converted int to String using valueOf: 100

Boolean value from String: true

Boolean true as primitive: true

Cannot convert null to Integer wrapper: java.lang.NumberFormatException: null

Boolean from null String: false

1. Write a program where a method accepts an integer parameter and tries to change its value. Print the value before and after the method call.

Program:

public class ChangePrimitive {

public static void changeValue(int num) {

num = 50; // change local copy

System.out.println("Inside method: " + num);

}

public static void main(String[] args) {

int x = 10;

System.out.println("Before method: " + x);

changeValue(x);

System.out.println("After method: " + x); // unchanged

}

}

1. Create a method that takes two integer values and swaps them. Show that the original values remain unchanged after the method call.

Program:

public class SwapPrimitive {

public static void swap(int a, int b) {

int temp = a;

a = b;

b = temp;

System.out.println("Inside method: a=" + a + ", b=" + b);

}

public static void main(String[] args) {

int x = 5, y = 10;

System.out.println("Before swap: x=" + x + ", y=" + y);

swap(x, y);

System.out.println("After swap: x=" + x + ", y=" + y); // unchanged

}

}

1. Write a Java program to pass primitive data types to a method and observe whether changes inside the method affect the original variables.

Program:

public class PrimitivePassDemo {

public static void modifyValues(int a, double b) {

a = 99;

b = 88.8;

System.out.println("Inside method: a=" + a + ", b=" + b);

}

public static void main(String[] args) {

int num = 10;

double val = 20.5;

System.out.println("Before: num=" + num + ", val=" + val);

modifyValues(num, val);

System.out.println("After: num=" + num + ", val=" + val); // unchanged

}

}

**Call by Reference (Using Objects)**

1. Create a class Box with a variable length. Write a method that modifies the value of length by passing the Box object. Show that the original object is modified.

Program:

class Box {

int length;

}

public class ModifyBox {

public static void changeLength(Box b) {

b.length = 50;

}

public static void main(String[] args) {

Box box = new Box();

box.length = 10;

System.out.println("Before: length=" + box.length);

changeLength(box);

System.out.println("After: length=" + box.length); // changed

}

}

1. Write a Java program to pass an object to a method and modify its internal fields. Verify that the changes reflect outside the method.

Program:

class Person {

String name;

}

public class ModifyPerson {

public static void changeName(Person p) {

p.name = "John";

}

public static void main(String[] args) {

Person person = new Person();

person.name = "Alice";

System.out.println("Before: " + person.name);

changeName(person);

System.out.println("After: " + person.name); // changed

}

}

1. Create a class Student with name and marks. Write a method to update the marks of a student. Demonstrate the changes in the original object.

Program:

class Student {

String name;

int marks;

}

public class UpdateStudent {

public static void updateMarks(Student s, int newMarks) {

s.marks = newMarks;

}

public static void main(String[] args) {

Student stu = new Student();

stu.name = "Ashwin";

stu.marks = 70;

System.out.println("Before: " + stu.name + " - " + stu.marks);

updateMarks(stu, 90);

System.out.println("After: " + stu.name + " - " + stu.marks);

}

}

1. Create a program to show that Java is strictly "call by value" even when passing objects (object references are passed by value).

Program:

class MyObj {

int data;

}

public class CallByValueObjects {

public static void reassign(MyObj obj) {

obj = new MyObj(); // new object, only local reference changes

obj.data = 500;

}

public static void main(String[] args) {

MyObj m = new MyObj();

m.data = 100;

System.out.println("Before: " + m.data);

reassign(m);

System.out.println("After: " + m.data); // unchanged

}

}

1. Write a program where you assign a new object to a reference passed into a method. Show that the original reference does not change.

Program:

class TestObj {

int val;

}

public class AssignNewObject {

public static void assign(TestObj t) {

t = new TestObj();

t.val = 999;

}

public static void main(String[] args) {

TestObj obj = new TestObj();

obj.val = 100;

System.out.println("Before: " + obj.val);

assign(obj);

System.out.println("After: " + obj.val); // unchanged

}

}

1. Explain the difference between passing primitive and non-primitive types to methods in Java with examples.

Program:  
public class PassDifference {

public static void changePrimitive(int a) {

a = 100; // local copy only

}

public static void changeObject(StringBuilder sb) {

sb.append(" World"); // modifies same object

}

public static void main(String[] args) {

int num = 10;

StringBuilder text = new StringBuilder("Hello");

changePrimitive(num);

changeObject(text);

System.out.println("Primitive after change: " + num); // unchanged

System.out.println("Object after change: " + text); // changed

}

}

1. Can you simulate call by reference in Java using a wrapper class or array? Justify with a program.

Program:

public class SimulateCallByRef {

public static void updateArray(int[] arr) {

arr[0] = 999; // modifies same array

}

public static void main(String[] args) {

int[] numbers = {10};

System.out.println("Before: " + numbers[0]);

updateArray(numbers);

System.out.println("After: " + numbers[0]); // changed

}

}

MultiThreading

1 Write a program to create a thread by extending the Thread class and print numbers from 1 to 5.

Program:

class MyThread extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println(i);

}

}

}

public class ThreadExtendDemo {

public static void main(String[] args) {

new MyThread().start();

}

}

2 Create a thread by implementing the Runnable interface that prints the current thread name.

Program:

class MyRunnable implements Runnable {

public void run() {

System.out.println("Current Thread: " + Thread.currentThread().getName());

}

}

public class RunnableDemo {

public static void main(String[] args) {

Thread t = new Thread(new MyRunnable());

t.start();

}

}

3 Write a program to create two threads, each printing a different message 5 times.

Program:

class MsgThread extends Thread {

String msg;

MsgThread(String msg) { this.msg = msg; }

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(msg);

}

}

}

public class TwoMessageThreads {

public static void main(String[] args) {

new MsgThread("Hello").start();

new MsgThread("World").start();

}

}

4 Demonstrate the use of Thread.sleep() by pausing execution between numbers from 1 to 3.

Program:

public class SleepDemo {

public static void main(String[] args) throws InterruptedException {

for (int i = 1; i <= 3; i++) {

System.out.println(i);

Thread.sleep(1000);

}

}

}

5 Create a thread and use Thread.yield() to pause and give chance to another thread.

Program:

class YieldThread extends Thread {

public void run() {

for (int i = 1; i <= 3; i++) {

System.out.println(getName() + " running");

Thread.yield();

}

}

}

public class YieldDemo {

public static void main(String[] args) {

new YieldThread().start();

new YieldThread().start();

}

}

6 Implement a program where two threads print even and odd numbers respectively.

Program:

class EvenThread extends Thread {

public void run() {

for (int i = 0; i <= 10; i += 2)

System.out.println("Even: " + i);

}

}

class OddThread extends Thread {

public void run() {

for (int i = 1; i <= 10; i += 2)

System.out.println("Odd: " + i);

}

}

public class EvenOddThreads {

public static void main(String[] args) {

new EvenThread().start();

new OddThread().start();

}

}

7 Create a program that starts three threads and sets different priorities for them.

Program:

class PriorityThread extends Thread {

public void run() {

System.out.println(getName() + " Priority: " + getPriority());

}

}

public class PriorityDemo {

public static void main(String[] args) {

Thread t1 = new PriorityThread();

Thread t2 = new PriorityThread();

Thread t3 = new PriorityThread();

t1.setPriority(Thread.MIN\_PRIORITY);

t2.setPriority(Thread.NORM\_PRIORITY);

t3.setPriority(Thread.MAX\_PRIORITY);

t1.start(); t2.start(); t3.start();

}

}

8 Write a program to demonstrate Thread.join() – wait for a thread to finish before proceeding.

Program:

class JoinThread extends Thread {

public void run() {

for (int i = 1; i <= 3; i++)

System.out.println(getName() + " " + i);

}

}

public class JoinDemo {

public static void main(String[] args) throws InterruptedException {

Thread t1 = new JoinThread();

t1.start();

t1.join();

System.out.println("Main thread continues after t1");

}

}

9 Show how to stop a thread using a boolean flag.

Program:

class StopThread extends Thread {

boolean running = true;

public void run() {

while (running) {

System.out.println("Running...");

}

}

}

public class StopFlagDemo {

public static void main(String[] args) throws InterruptedException {

StopThread t = new StopThread();

t.start();

Thread.sleep(100);

t.running = false;

}

}

10 Create a program with multiple threads that access a shared counter without synchronization. Show the race condition.

Program:

class Counter {

int count = 0;

void increment() { count++; }

}

public class RaceConditionDemo {

public static void main(String[] args) throws InterruptedException {

Counter c = new Counter();

Runnable task = () -> {

for (int i = 0; i < 1000; i++) c.increment();

};

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

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

t1.join(); t2.join();

System.out.println("Final count: " + c.count);

}

}

11 Solve the above problem using synchronized keyword to prevent race condition.

Program:

class SafeCounter {

int count = 0;

synchronized void increment() { count++; }

}

public class SynchronizedCounterDemo {

public static void main(String[] args) throws InterruptedException {

SafeCounter c = new SafeCounter();

Runnable task = () -> {

for (int i = 0; i < 1000; i++) c.increment();

};

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

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

t1.join(); t2.join();

System.out.println("Final count: " + c.count);

}

}

12 Write a Java program using synchronized block to ensure mutual exclusion.

Program:

class BlockCounter {

int count = 0;

void increment() {

synchronized (this) { count++; }

}

}

public class SyncBlockDemo {

public static void main(String[] args) throws InterruptedException {

BlockCounter c = new BlockCounter();

Runnable task = () -> {

for (int i = 0; i < 1000; i++) c.increment();

};

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

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

t1.join(); t2.join();

System.out.println("Final count: " + c.count);

}

}

13 Implement a BankAccount class accessed by multiple threads to deposit and withdraw money. Use synchronization.

Program:

class BankAccount {

private int balance = 1000;

synchronized void deposit(int amount) {

balance += amount;

System.out.println("Deposited " + amount + ", Balance: " + balance);

}

synchronized void withdraw(int amount) {

if (balance >= amount) {

balance -= amount;

System.out.println("Withdrew " + amount + ", Balance: " + balance);

} else {

System.out.println("Insufficient funds");

}

}

}

public class BankDemo {

public static void main(String[] args) {

BankAccount account = new BankAccount();

new Thread(() -> account.deposit(500)).start();

new Thread(() -> account.withdraw(200)).start();

}

}

14 Create a Producer-Consumer problem using wait() and notify().

Program:

import java.util.LinkedList;

class Shared {

LinkedList<Integer> list = new LinkedList<>();

int capacity = 2;

public synchronized void produce(int value) throws InterruptedException {

while (list.size() == capacity) wait();

list.add(value);

System.out.println("Produced: " + value);

notify();

}

public synchronized void consume() throws InterruptedException {

while (list.isEmpty()) wait();

int val = list.removeFirst();

System.out.println("Consumed: " + val);

notify();

}

}

public class ProducerConsumerDemo {

public static void main(String[] args) {

Shared s = new Shared();

new Thread(() -> {

for (int i = 1; i <= 5; i++) {

try { s.produce(i); } catch (Exception e) {}

}

}).start();

new Thread(() -> {

for (int i = 1; i <= 5; i++) {

try { s.consume(); } catch (Exception e) {}

}

}).start();

}

}

15 Create a program where one thread prints A-Z and another prints 1-26 alternately.

Program:

class Alternator {

boolean letterTurn = true;

public synchronized void printLetter(char letter) {

while (!letterTurn) {

try { wait(); } catch (InterruptedException e) {}

}

System.out.println(letter);

letterTurn = false;

notifyAll();

}

public synchronized void printNumber(int num) {

while (letterTurn) {

try { wait(); } catch (InterruptedException e) {}

}

System.out.println(num);

letterTurn = true;

notifyAll();

}

}

public class AlternatePrint {

public static void main(String[] args) {

Alternator alt = new Alternator();

Thread t1 = new Thread(() -> {

for (char c = 'A'; c <= 'Z'; c++) {

alt.printLetter(c);

}

});

Thread t2 = new Thread(() -> {

for (int i = 1; i <= 26; i++) {

alt.printNumber(i);

}

});

t1.start();

t2.start();

}

}

16 Write a program that demonstrates inter-thread communication using wait() and notifyAll().

Program:

class SharedResource {

private boolean available = false;

public synchronized void produce() {

available = true;

System.out.println("Produced item");

notifyAll();

}

public synchronized void consume() {

while (!available) {

try { wait(); } catch (InterruptedException e) {}

}

System.out.println("Consumed item");

available = false;

}

}

public class WaitNotifyAllDemo {

public static void main(String[] args) {

SharedResource resource = new SharedResource();

Thread producer = new Thread(() -> {

try { Thread.sleep(1000); } catch (Exception e) {}

resource.produce();

});

Thread consumer1 = new Thread(resource::consume);

Thread consumer2 = new Thread(resource::consume);

consumer1.start();

consumer2.start();

producer.start();

}

}

17 Create a daemon thread that runs in background and prints time every second.

import java.time.LocalTime;

public class DaemonThreadDemo {

public static void main(String[] args) {

Thread timeThread = new Thread(() -> {

while (true) {

System.out.println("Time: " + LocalTime.now());

try { Thread.sleep(1000); } catch (InterruptedException e) {}

}

});

timeThread.setDaemon(true);

timeThread.start();

try { Thread.sleep(5000); } catch (InterruptedException e) {}

System.out.println("Main thread exiting");

}

}

18 Demonstrate the use of Thread.isAlive() to check thread status.

public class IsAliveDemo {

public static void main(String[] args) throws InterruptedException {

Thread t = new Thread(() -> {

System.out.println("Thread running...");

});

System.out.println("Before start: " + t.isAlive());

t.start();

System.out.println("After start: " + t.isAlive());

t.join();

System.out.println("After join: " + t.isAlive());

}

}

19 Write a program to demonstrate thread group creation and management.

Program:

public class ThreadGroupDemo {

public static void main(String[] args) {

ThreadGroup group = new ThreadGroup("MyGroup");

Runnable task = () -> {

System.out.println(Thread.currentThread().getName() + " in " + Thread.currentThread().getThreadGroup().getName());

};

Thread t1 = new Thread(group, task, "Thread-1");

Thread t2 = new Thread(group, task, "Thread-2");

t1.start();

t2.start();

System.out.println("Active threads in group: " + group.activeCount());

}

}

20 Create a thread that performs a simple task (like multiplication) and returns result using Callable and Future.

Program:

import java.util.concurrent.\*;

class MultiplyTask implements Callable<Integer> {

private int a, b;

MultiplyTask(int a, int b) { this.a = a; this.b = b; }

public Integer call() {

return a \* b;

}

}

public class CallableFutureDemo {

public static void main(String[] args) throws Exception {

ExecutorService executor = Executors.newSingleThreadExecutor();

Future<Integer> result = executor.submit(new MultiplyTask(5, 6));

System.out.println("Multiplication result: " + result.get());

executor.shutdown();

}

}