# Day - 7

1. Check if character is a Digit

Code:

public class CheckDigit {  
 public static void main(String[] args) {  
 char ch = '5';  
 if (Character.isDigit(ch)) {  
 System.out.println(ch + " is a digit.");  
 } else {  
 System.out.println(ch + " is not a digit.");  
 }  
 }  
}

2. Compare two Strings

Code:

public class CompareStrings {  
 public static void main(String[] args) {  
 String s1 = "Hello";  
 String s2 = "World";  
 System.out.println("Comparison result: " + s1.compareTo(s2));  
 }  
}

3. Convert using valueOf method

Code:

public class ValueOfExample {  
 public static void main(String[] args) {  
 int num = 100;  
 String str = String.valueOf(num);  
 System.out.println("String value: " + str);  
 }  
}

4. Create Boolean Wrapper usage

Code:

public class BooleanWrapper {  
 public static void main(String[] args) {  
 Boolean boolObj = Boolean.valueOf(true);  
 System.out.println("Boolean value: " + boolObj);  
 }  
}

5. Convert null to wrapper classes

Code:  
public class NullToWrapper {  
 public static void main(String[] args) {  
 Integer num = null;  
 System.out.println("Integer value: " + num);  
 }  
}

## 2. Pass by Value

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.

Code:

public class PassByValue1 {  
 public static void changeValue(int x) {  
 x = 50;  
 System.out.println("Inside method: " + x);  
 }  
 public static void main(String[] args) {  
 int num = 10;  
 System.out.println("Before: " + num);  
 changeValue(num);  
 System.out.println("After: " + num);  
 }  
}

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

Code:  
public class SwapIntegers {  
 public static void swap(int a, int b) {  
 int temp = a;  
 a = b;  
 b = temp;  
 System.out.println("Inside swap: 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);  
 }  
}

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

Code:

public class PrimitivePass {  
 public static void modify(int num) {  
 num += 10;  
 System.out.println("Inside method: " + num);  
 }  
 public static void main(String[] args) {  
 int val = 20;  
 System.out.println("Before: " + val);  
 modify(val);  
 System.out.println("After: " + val);  
 }  
}

## 3. Call by Reference (Using Objects)

4.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.

Code:

class Box {  
 int length;  
}  
public class CallByReference {  
 public static void modify(Box b) {  
 b.length = 50;  
 }  
 public static void main(String[] args) {  
 Box b = new Box();  
 b.length = 10;  
 System.out.println("Before: " + b.length);  
 modify(b);  
 System.out.println("After: " + b.length);  
 }  
}

5.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.

Code:

class Person {  
 String name;  
}  
public class ModifyObject {  
 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);  
 }  
}

6.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.

Code:

class Student {  
 String name;  
 int marks;  
}  
public class UpdateMarks {  
 public static void update(Student s, int newMarks) {  
 s.marks = newMarks;  
 }  
 public static void main(String[] args) {  
 Student st = new Student();  
 st.name = "Ravi";  
 st.marks = 50;  
 System.out.println("Before: " + st.marks);  
 update(st, 90);  
 System.out.println("After: " + st.marks);  
 }  
}

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

Code:

class MyObj {  
 int val;  
}  
public class CallByValueObjects {  
 public static void change(MyObj obj) {  
 obj = new MyObj();  
 obj.val = 100;  
 }  
 public static void main(String[] args) {  
 MyObj o = new MyObj();  
 o.val = 50;  
 change(o);  
 System.out.println("After: " + o.val);  
 }  
}

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

Code:

class Car {  
 String model;  
}  
public class ChangeReference {  
 public static void change(Car c) {  
 c = new Car();  
 c.model = "Tesla";  
 }  
 public static void main(String[] args) {  
 Car car = new Car();  
 car.model = "BMW";  
 change(car);  
 System.out.println("After: " + car.model);  
 }  
}

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

Code:

public class PrimitiveVsObject {  
 public static void modifyInt(int x) { x = 100; }  
 public static void modifyArray(int[] arr) { arr[0] = 100; }  
 public static void main(String[] args) {  
 int num = 10;  
 int[] arr = {1, 2, 3};  
 modifyInt(num);  
 modifyArray(arr);  
 System.out.println("Primitive after: " + num);  
 System.out.println("Array after: " + arr[0]);  
 }  
}

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

Code:

public class CallByReferenceSim {  
 public static void modify(int[] arr) {  
 arr[0] = 500;  
 }  
 public static void main(String[] args) {  
 int[] num = {100};  
 System.out.println("Before: " + num[0]);  
 modify(num);  
 System.out.println("After: " + num[0]);  
 }  
}

## 4. MultiThreading Programs

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

Code:

class MyThread1 extends Thread {  
 public void run() {  
 for (int i = 1; i <= 5; i++) {  
 System.out.println(i);  
 }  
 }  
}  
public class ThreadExtendExample {  
 public static void main(String[] args) {  
 MyThread1 t = new MyThread1();  
 t.start();  
 }  
}

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

Code:

class MyRunnable implements Runnable {  
 public void run() {  
 System.out.println("Current Thread: " + Thread.currentThread().getName());  
 }  
}  
public class RunnableExample {  
 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.

Code:

class MessageThread extends Thread {  
 String message;  
 MessageThread(String msg) { message = msg; }  
 public void run() {  
 for (int i = 0; i < 5; i++) {  
 System.out.println(message);  
 }  
 }  
}  
public class TwoThreadsExample {  
 public static void main(String[] args) {  
 new MessageThread("Hello").start();  
 new MessageThread("World").start();  
 }  
}

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

Code:

public class SleepExample {  
 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.

Code:

class YieldThread extends Thread {  
 public void run() {  
 for (int i = 1; i <= 3; i++) {  
 System.out.println(getName() + " running");  
 Thread.yield();  
 }  
 }  
}  
public class YieldExample {  
 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.

Code:

class EvenThread extends Thread {  
 public void run() {  
 for (int i = 2; i <= 10; i += 2) {  
 System.out.println("Even: " + i);  
 }  
 }  
}  
class OddThread extends Thread {  
 public void run() {  
 for (int i = 1; i <= 9; 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.

Code:

class PriorityThread extends Thread {  
 public void run() {  
 System.out.println(getName() + " Priority: " + getPriority());  
 }  
}  
public class PriorityExample {  
 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.

Code:

class JoinThread extends Thread {  
 public void run() {  
 for (int i = 1; i <= 3; i++) {  
 System.out.println(getName() + ": " + i);  
 }  
 }  
}  
public class JoinExample {  
 public static void main(String[] args) throws InterruptedException {  
 JoinThread t1 = new JoinThread();  
 t1.start();  
 t1.join();  
 System.out.println("Main thread after join");  
 }  
}

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

Code:

class StopThread extends Thread {  
 volatile boolean running = true;  
 public void run() {  
 while (running) {  
 System.out.println("Running...");  
 }  
 }  
}  
public class StopFlagExample {  
 public static void main(String[] args) throws InterruptedException {  
 StopThread t = new StopThread();  
 t.start();  
 Thread.sleep(1000);  
 t.running = false;  
 }  
}

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

Code:

class Counter {  
 int count = 0;  
 public void increment() { count++; }  
}  
public class RaceCondition {  
 public static void main(String[] args) throws InterruptedException {  
 Counter c = new Counter();  
 Runnable r = () -> { for (int i = 0; i < 1000; i++) c.increment(); };  
 Thread t1 = new Thread(r);  
 Thread t2 = new Thread(r);  
 t1.start(); t2.start();  
 t1.join(); t2.join();  
 System.out.println("Count: " + c.count);  
 }  
}

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

Code:  
class SyncCounter {  
 int count = 0;  
 public synchronized void increment() { count++; }  
}  
public class SyncExample {  
 public static void main(String[] args) throws InterruptedException {  
 SyncCounter c = new SyncCounter();  
 Runnable r = () -> { for (int i = 0; i < 1000; i++) c.increment(); };  
 Thread t1 = new Thread(r);  
 Thread t2 = new Thread(r);  
 t1.start(); t2.start();  
 t1.join(); t2.join();  
 System.out.println("Count: " + c.count);  
 }  
}

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

Code:

class BlockCounter {  
 int count = 0;  
 public void increment() {  
 synchronized (this) { count++; }  
 }  
}  
public class SyncBlockExample {  
 public static void main(String[] args) throws InterruptedException {  
 BlockCounter c = new BlockCounter();  
 Runnable r = () -> { for (int i = 0; i < 1000; i++) c.increment(); };  
 Thread t1 = new Thread(r);  
 Thread t2 = new Thread(r);  
 t1.start(); t2.start();  
 t1.join(); t2.join();  
 System.out.println("Count: " + c.count);  
 }  
}

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

Code:

class BankAccount {  
 private int balance = 1000;  
 public synchronized void deposit(int amount) {  
 balance += amount;  
 System.out.println("Deposited: " + amount + ", Balance: " + balance);  
 }  
 public synchronized void withdraw(int amount) {  
 if (balance >= amount) {  
 balance -= amount;  
 System.out.println("Withdrawn: " + amount + ", Balance: " + balance);  
 } else {  
 System.out.println("Insufficient balance");  
 }  
 }  
}  
public class BankExample {  
 public static void main(String[] args) {  
 BankAccount acc = new BankAccount();  
 Thread t1 = new Thread(() -> acc.deposit(500));  
 Thread t2 = new Thread(() -> acc.withdraw(300));  
 t1.start(); t2.start();  
 }  
}

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

Code:

class SharedBuffer {  
 int data;  
 boolean hasData = false;  
 public synchronized void produce(int value) throws InterruptedException {  
 while (hasData) wait();  
 data = value;  
 hasData = true;  
 System.out.println("Produced: " + value);  
 notify();  
 }  
 public synchronized void consume() throws InterruptedException {  
 while (!hasData) wait();  
 System.out.println("Consumed: " + data);  
 hasData = false;  
 notify();  
 }  
}  
public class ProducerConsumer {  
 public static void main(String[] args) {  
 SharedBuffer buffer = new SharedBuffer();  
 new Thread(() -> {  
 try { for (int i = 1; i <= 5; i++) buffer.produce(i); } catch (Exception e) {}  
 }).start();  
 new Thread(() -> {  
 try { for (int i = 1; i <= 5; i++) buffer.consume(); } catch (Exception e) {}  
 }).start();  
 }  
}

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

Code:

class AlternatePrinter {  
 boolean letterTurn = true;  
 public synchronized void printLetter(char letter) throws InterruptedException {  
 while (!letterTurn) wait();  
 System.out.print(letter + " ");  
 letterTurn = false;  
 notify();  
 }  
 public synchronized void printNumber(int num) throws InterruptedException {  
 while (letterTurn) wait();  
 System.out.print(num + " ");  
 letterTurn = true;  
 notify();  
 }  
}  
public class AlternateExample {  
 public static void main(String[] args) {  
 AlternatePrinter printer = new AlternatePrinter();  
 Thread letters = new Thread(() -> {  
 try { for (char c = 'A'; c <= 'Z'; c++) printer.printLetter(c); } catch (Exception e) {}  
 });  
 Thread numbers = new Thread(() -> {  
 try { for (int i = 1; i <= 26; i++) printer.printNumber(i); } catch (Exception e) {}  
 });  
 letters.start();  
 numbers.start();  
 }  
}

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

Code:

class SharedData {  
 public synchronized void display(String msg) {  
 System.out.println(msg);  
 notifyAll();  
 }  
}  
public class NotifyAllExample {  
 public static void main(String[] args) {  
 SharedData data = new SharedData();  
 Runnable r = () -> { synchronized (data) { try { data.wait(); } catch (Exception e) {} System.out.println(Thread.currentThread().getName() + " notified"); } };  
 new Thread(r, "T1").start();  
 new Thread(r, "T2").start();  
 new Thread(() -> data.display("Notifying all")).start();  
 }  
}

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

Code:

import java.time.LocalTime;  
public class DaemonExample {  
 public static void main(String[] args) {  
 Thread t = new Thread(() -> {  
 while (true) {  
 System.out.println(LocalTime.now());  
 try { Thread.sleep(1000); } catch (Exception e) {}  
 }  
 });  
 t.setDaemon(true);  
 t.start();  
 try { Thread.sleep(5000); } catch (Exception e) {}  
 }  
}

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

Code:

public class IsAliveExample {  
 public static void main(String[] args) throws InterruptedException {  
 Thread t = new Thread(() -> System.out.println("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.

Code:

public class ThreadGroupExample {  
 public static void main(String[] args) {  
 ThreadGroup group = new ThreadGroup("MyGroup");  
 Runnable r = () -> System.out.println(Thread.currentThread().getName());  
 new Thread(group, r, "T1").start();  
 new Thread(group, r, "T2").start();  
 group.list();  
 }  
}

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

Code:

import java.util.concurrent.\*;  
public class CallableExample {  
 public static void main(String[] args) throws Exception {  
 Callable<Integer> task = () -> 5 \* 10;  
 ExecutorService executor = Executors.newSingleThreadExecutor();  
 Future<Integer> future = executor.submit(task);  
 System.out.println("Result: " + future.get());  
 executor.shutdown();  
 }  
}