Wrapper classes

1. Check if character is a Digit

Answer

```
public class DigitCheck {
  public static void main(String[] args) {
    char c = '5';
    System.out.println(Character.isDigit(c) ? c + " is a digit" : c + " is not a digit");
  }
}
```

2. Compare two Strings

Answer

```
public class StringCompare {
  public static void main(String[] args) {
    String s1 = "Hello";
    String s2 = "World";
    System.out.println(s1.equals(s2) ? "Equal" : "Not Equal");
  }
}
```

3. Convert using valueof method

Answer

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

4. Create Boolean Wrapper usage

Answer

```
public class BooleanWrapper {
    public static void main(String[] args) {
        Boolean b1 = Boolean.valueOf("true");
        Boolean b2 = Boolean.valueOf("false");
        System.out.println("b1: " + b1 + ", b2: " + b2);
    }
}
```

5. Convert null to wrapper classes

Answer

```
     public class NullWrapper {
     public static void main(String[] args) {
     String s = null;
      Integer num = (s == null) ? null : Integer.valueOf(s);
      System.out.println("Converted null to wrapper: " + num);
    }
}
```

Pass by value and pass by reference

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.

Answer

```
public class PassByValueDemo1 {
  public static void changeValue(int x) {
    x = 100;
  }

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

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

Answer

```
public class PassByValueDemo2 {
  public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    System.out.println("Inside method after swap: a=" + a + ", b=" + b);
  }

  public static void main(String[] args) {
    int x = 10, y = 20;
    System.out.println("Before method call: x=" + x + ", y=" + y);
    swap(x, y);
    System.out.println("After method call: 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.

Answer

```
public class PrimitiveDemo {
  public static void modify(int num) {
    num = num + 10;
  }

  public static void main(String[] args) {
    int a = 5;
    System.out.println("Before method call: " + a);
    modify(a);
    System.out.println("After method call: " + a);
  }
}
```

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.

Answer

```
class Box {
  int length;
}

public class BoxDemo {
  public static void changeLength(Box b) {
    b.length = 50;
  }

  public static void main(String[] args) {
    Box b1 = new Box();
    b1.length = 10;
    System.out.println("Before: " + b1.length);
    changeLength(b1);
    System.out.println("After: " + b1.length); // modified
  }
}
```

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

Answer

```
class Car {
  String color;
}

public class ObjectModifyDemo {
  public static void paint(Car c) {
    c.color = "Red";
  }

  public static void main(String[] args) {
    Car myCar = new Car();
    myCar.color = "Blue";
    System.out.println("Before: " + myCar.color);
    paint(myCar);
    System.out.println("After: " + myCar.color); // changed
  }
}
```

---

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

Answer

```
class Person {
  String name;
}

public class CallByValueDemo {
  public static void change(Person p) {
    p = new Person(); // new reference
    p.name = "Changed";
  }

  public static void main(String[] args) {
    Person p1 = new Person();
    p1.name = "Original";
    System.out.println("Before: " + p1.name);
    change(p1);
    System.out.println("After: " + p1.name); // remains Original
  }
}
```

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.

Answer

```
class Dog {
  String name;
}

public class ObjectReferenceDemo {
  public static void assignNew(Dog d) {
    d = new Dog();
    d.name = "Max";
  }

  public static void main(String[] args) {
    Dog dog1 = new Dog();
    dog1.name = "Buddy";
    System.out.println("Before: " + dog1.name);
    assignNew(dog1);
    System.out.println("After: " + dog1.name); // unchanged
  }
}
```

---

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

Answer

□ **Primitive:** Passed by value → changes inside method don't affect original.

□ **Objects:** Reference is passed by value → modifying fields affects original, but assigning new object doesn't.

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

Answer public class WrapperDemo {

```
  public static void modify(int[] arr) {
    arr[0] = 99;
  }

  public static void main(String[] args) {
    int[] nums = {10};
    System.out.println("Before: " + nums[0]);
    modify(nums);
    System.out.println("After: " + nums[0]); // changed
  }
}
```

MultiThreading

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

Answer

```
class MyThread1 extends Thread {
  public void run() {
    for (int i = 1; i <= 5; i++) {
      System.out.println(i);
    }
  }
}

public class ThreadDemo1 {
  public static void main(String[] args) {
    MyThread1 t = new MyThread1();
    t.start();
  }
}
```

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

Answer

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

public class ThreadDemo2 {
  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.

Answer

```
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 ThreadDemo3 {
  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.

Answer

```
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); // pause 1 sec
    }
  }
}
```

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

Answer

```
class YieldThread extends Thread {
  public void run() {
    for (int i = 0; i < 5; 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.

Answer

```
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 EvenOddDemo {
  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.

Answer

```
class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(getName() + " is running");
        }
    }
}

public class PriorityDemo {
    public static void main(String[] args) {
        MyThread t1 = new MyThread("Thread-1");
        MyThread t2 = new MyThread("Thread-2");
        MyThread t3 = new MyThread("Thread-3");

        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.

Answer

```
class Worker extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Worker: " + i);
        }
    }
}

public class JoinDemo {
    public static void main(String[] args) throws InterruptedException {
        Worker t = new Worker();
        t.start();
        t.join();  // wait until t finishes
        System.out.println("Main thread continues after Worker finishes.");
    }
}
```

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

Answer

```
class StopThread extends Thread {
    private volatile boolean running = true;

    public void run() {
        while (running) {
            System.out.println("Thread running...");
        }
        System.out.println("Thread stopped.");
    }

    public void stopThread() {
        running = false;
    }
}

public class StopFlagDemo {
    public static void main(String[] args) throws InterruptedException {
        StopThread t = new StopThread();
        t.start();
        Thread.sleep(1000);
        t.stopThread();
    }
}
```

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

Answer

```
class Counter {
    int count = 0;
    public void increment() {
        count++;
    }
}
```

```
public class RaceConditionDemo {
    public static void main(String[] args) throws InterruptedException {
        Counter c = new Counter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

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

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

        System.out.println("Final Count (should be 2000): " + c.count);
    }
}
```

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

Answer

```
class SafeCounter {
    int count = 0;

    public synchronized void increment() {
        count++;
    }
}

public class SyncDemo {
    public static void main(String[] args) throws InterruptedException {
        SafeCounter c = new SafeCounter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

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

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

        System.out.println("Final Count (safe): " + c.count);
    }
}
```

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

Answer

```
class SyncBlockCounter {
    int count = 0;
    public void increment() {
        synchronized (this) {
            count++;
        }
    }
}

public class SyncBlockDemo {
    public static void main(String[] args) throws InterruptedException {
        SyncBlockCounter c = new SyncBlockCounter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) c.increment();
        });

        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.

Answer

```
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 BankDemo {
  public static void main(String[] args) {
    BankAccount account = new BankAccount();

    Thread t1 = new Thread(() -> account.deposit(500));
    Thread t2 = new Thread(() -> account.withdraw(700));
```

```
        t1.start();
        t2.start();
    }
}
```

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

Answer

```
import java.util.LinkedList;
import java.util.Queue;

class BoundedBuffer {
    private final Queue<Integer> q = new LinkedList<>();
    private final int capacity;

    BoundedBuffer(int capacity) { this.capacity = capacity; }

    public synchronized void produce(int item) throws InterruptedException {
        while (q.size() == capacity) wait();
        q.add(item);
        System.out.println("Produced: " + item + " | size=" + q.size());
        notifyAll(); // wake consumers
    }

    public synchronized int consume() throws InterruptedException {
        while (q.isEmpty()) wait();
        int val = q.remove();
        System.out.println("  Consumed: " + val + " | size=" + q.size());
        notifyAll(); // wake producers
        return val;
    }
}

public class ProducerConsumerDemo {
    public static void main(String[] args) {
        BoundedBuffer buffer = new BoundedBuffer(5);

        Thread producer = new Thread(() -> {
            for (int i = 1; i <= 20; i++) {
                try { buffer.produce(i); Thread.sleep(50); } catch (InterruptedException ignored) {}
            }
        }, "Producer");

        Thread consumer = new Thread(() -> {
            for (int i = 1; i <= 20; i++) {
                try { buffer.consume(); Thread.sleep(100); } catch (InterruptedException ignored) {}
            }
        }, "Consumer");

        producer.start();
        consumer.start();
    }
}
```

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

Answer

```
class Alternator {
    private boolean letterTurn = true;
    public synchronized void printLetter(char c) throws InterruptedException {
        while (!letterTurn) wait();
        System.out.print(c + " ");
        letterTurn = false;
        notifyAll();
    }

    public synchronized void printNumber(int n) throws InterruptedException {
        while (letterTurn) wait();
        System.out.print(n + " ");
        letterTurn = true;
        notifyAll();
    }
}

public class AlternateAZ12 {
    public static void main(String[] args) {
        Alternator alt = new Alternator();

        Thread letters = new Thread(() -> {
            for (char c = 'A'; c <= 'Z'; c++) {
                try { alt.printLetter(c); } catch (InterruptedException ignored) {}
            }
        }, "Letters");

        Thread numbers = new Thread(() -> {
            for (int i = 1; i <= 26; i++) {
                try { alt.printNumber(i); } catch (InterruptedException ignored) {}
            }
        }, "Numbers");

        letters.start();
        numbers.start();
    }
}
```

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

Answer

```
class StartGate {
  private boolean open = false;
  public synchronized void await() throws InterruptedException {
    while (!open) wait();
  }
  public synchronized void open() {
    open = true;
    notifyAll();
  }
}

public class NotifyAllDemo {
  public static void main(String[] args) throws InterruptedException {
    StartGate gate = new StartGate();

    Runnable worker = () -> {
      try {
        System.out.println(Thread.currentThread().getName() + " waiting");
        gate.await();
        System.out.println(Thread.currentThread().getName() + " started work");
      } catch (InterruptedException ignored) {}
    };

    Thread w1 = new Thread(worker, "Worker-1");
    Thread w2 = new Thread(worker, "Worker-2");
    Thread w3 = new Thread(worker, "Worker-3");

    w1.start(); w2.start(); w3.start();

    Thread.sleep(800);
    System.out.println("Main opening gate...");
    gate.open();
  }
}
```

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

Answer

```
import java.time.LocalTime;

public class DaemonTimePrinter {
  public static void main(String[] args) throws InterruptedException {
    Thread daemon = new Thread(() -> {
      while (true) {
        System.out.println("Time: " + LocalTime.now());
        try { Thread.sleep(1000); } catch (InterruptedException ignored) {}
      }
    }, "Clock");
    daemon.setDaemon(true); // background
    daemon.start();
```

```
        Thread.sleep(3500);
        System.out.println("Main done; daemon will terminate when JVM exits.");
    }
}
```

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

Answer

```
public class IsAliveDemo {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> {
            for (int i = 1; i <= 3; i++) {
                System.out.println("Work " + i);
                try { Thread.sleep(300); } catch (InterruptedException ignored) {}
            }
        }, "Worker");

        System.out.println("Before start: isAlive=" + t.isAlive());
        t.start();
        System.out.println("After start: isAlive=" + t.isAlive());
        t.join();
        System.out.println("After join: isAlive=" + t.isAlive());
    }
}
```

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

Answer

```
public class ThreadGroupDemo {
  public static void main(String[] args) throws InterruptedException {
    ThreadGroup group = new ThreadGroup("MyGroup");

    Runnable job = () -> {
      try {
        while (!Thread.currentThread().isInterrupted()) {
          System.out.println(Thread.currentThread().getName() + " working");
          Thread.sleep(200);
        }
      } catch (InterruptedException ignored) { /* exit */ }
    };

    Thread t1 = new Thread(group, job, "T1");
    Thread t2 = new Thread(group, job, "T2");
    Thread t3 = new Thread(group, job, "T3");

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

    Thread.sleep(700);

    System.out.println("Active threads in group: " + group.activeCount());
    group.interrupt(); // interrupt all in the group

    t1.join(); t2.join(); t3.join();
    System.out.println("All threads stopped.");
  }
}
```

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

Answer

```
import java.util.concurrent.*;


class Multiplier implements Callable<Integer> {
  private final int a, b;
  Multiplier(int a, int b) { this.a = a; this.b = b; }
  @Override public Integer call() throws Exception {
    Thread.sleep(500); // simulate work
    return a * b;
  }
}


public class CallableFutureDemo {
  public static void main(String[] args) throws Exception {
    ExecutorService pool = Executors.newSingleThreadExecutor();
    Future<Integer> result = pool.submit(new Multiplier(12, 7));


    System.out.println("Doing other work in main...");
    Integer value = result.get(); // waits for result
    System.out.println("Result from Callable: " + value);


    pool.shutdown();
  }
}
```