Wrapper Classes

1. Write a Program for Check if character is a Digit

```
public class CheckDigit {
    public static void main(String[] args) {
        char ch = '5';
        System.out.println(Character.isDigit(ch));
    }
}
```

Output:

```
true
```

---

2. Write a Program for Compare two Strings

```
public class CompareStrings {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = "World";
        System.out.println(s1.equals(s2));
    }
}
```

Output:

```
false
```

---

3. Write a Program for Convert using valueOf method

```
public class ValueOfExample {
    public static void main(String[] args) {
        int num = 100;
        String str = String.valueOf(num);
        System.out.println(str + 50);
    }
}
```

Output:

```
10050
```

---

4. Write a Program for Create Boolean Wrapper usage

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

Output:

```
true
```

---

5. Write a Program for Convert null to wrapper classes

```
public class NullWrapper {
    public static void main(String[] args) {
        Integer i = null;
        System.out.println(i);
    }
}
```

Output:

```
null
```

---

1. Write a Program for Method accepts integer and tries to change value

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

Output:

```
Before: 10
After: 10
```

---

2. Write a Program for Swap two integers

```
public class SwapPrimitives {
    public static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
    }
    public static void main(String[] args) {
        int x = 5, y = 10;
        System.out.println("Before: x=" + x + " y=" + y);
        swap(x, y);
        System.out.println("After: x=" + x + " y=" + y);
    }
}
```

Output:

```
Before: x=5 y=10
After: x=5 y=10
```

---

3. Write a Program for Pass primitive data types

```
public class PrimitiveChange {
    public static void modify(int val) {
        val += 100;
    }
    public static void main(String[] args) {
        int n = 20;
        System.out.println("Before: " + n);
        modify(n);
        System.out.println("After: " + n);
    }
}
```

Output:

```
Before: 20
After: 20
```

---

4. Write a Program for Modify object field

```
class Box {
    int length;
}
public class ModifyObject {
    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: " + box.length);
        changeLength(box);
        System.out.println("After: " + box.length);
    }
}
```

Output:

```
Before: 10
After: 50
```

---

5. Write a Program for Pass object and modify field

```
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);
    }
}
```

Output:

```
Before: Alice
After: John
```

---

6. Write a Program for Update marks of Student

```
class Student {
    String name;
    int marks;
}
public class UpdateStudent {
    public static void updateMarks(Student s) {
        s.marks = 95;
    }
    public static void main(String[] args) {
        Student st = new Student();
        st.name = "Likitha";
        st.marks = 80;
        System.out.println("Before: " + st.marks);
        updateMarks(st);
        System.out.println("After: " + st.marks);
    }
}
```

Output:

```
Before: 80
After: 95
```

Pass-by-Value with Objects

7. Write a Program for Show Java is strictly call-by-value (object references passed by value)

```
class MyClass {
    int data;
}
public class CallByValueObject {
    public static void modify(MyClass obj) {
        obj.data = 100;
    }
    public static void main(String[] args) {
        MyClass mc = new MyClass();
        mc.data = 10;
        System.out.println("Before: " + mc.data);
        modify(mc);
        System.out.println("After: " + mc.data);
    }
}
```

Output:

```
Before: 10
After: 100
```

---

8. Write a Program for Assign new object to reference inside method

```
class Sample {
    int value;
}
```

```
public class NewObjectAssignment {
    public static void changeReference(Sample s) {
        s = new Sample();
        s.value = 500;
    }
    public static void main(String[] args) {
        Sample obj = new Sample();
        obj.value = 100;
        System.out.println("Before: " + obj.value);
        changeReference(obj);
        System.out.println("After: " + obj.value);
    }
}
```

Output:

```
Before: 100
After: 100
```

---

9. Write a Program for Difference between passing primitive and non-primitive types

```
class DataHolder {
    int num;
}
public class PrimitiveVsObject {
    public static void changePrimitive(int a) {
        a = 99;
    }
    public static void changeObject(DataHolder dh) {
        dh.num = 99;
    }
    public static void main(String[] args) {
        int x = 10;
        DataHolder holder = new DataHolder();
        holder.num = 10;
        changePrimitive(x);
        changeObject(holder);
        System.out.println("Primitive after method: " + x);
        System.out.println("Object after method: " + holder.num);
    }
}
```

Output:

```
Primitive after method: 10
Object after method: 99
```

---

10. Write a Program for Simulate call-by-reference using array

```
public class CallByReferenceSim {
    public static void modifyArray(int[] arr) {
        arr[0] = 999;
    }
    public static void main(String[] args) {
        int[] nums = {10};
        System.out.println("Before: " + nums[0]);
        modifyArray(nums);
        System.out.println("After: " + nums[0]);
    }
}
```

Output:

```
Before: 10
After: 999
```

Multithreading

1. Write a Program for Create thread by extending Thread class (1–5)

```
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++)
```

```
            System.out.println(i);
    }
}
public class ThreadExample1 {
    public static void main(String[] args) {
        new MyThread().start();
    }
}
```

Possible Output:

```
1
2
3
4
5
```

---

2. Write a Program for Create thread by implementing Runnable (print thread name)

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
}
public class ThreadExample2 {
    public static void main(String[] args) {
        new Thread(new MyRunnable()).start();
    }
}
```

Possible Output:

```
Thread-0
```

---

3. Write a Program for Two threads printing different messages 5 times

```
class MessageThread extends Thread {
    String msg;
    MessageThread(String m) { msg = m; }
    public void run() {
        for (int i = 0; i < 5; i++)
            System.out.println(msg);
    }
}
public class ThreadExample3 {
    public static void main(String[] args) {
        new MessageThread("Hello").start();
        new MessageThread("World").start();
    }
}
```

Possible Output (order may vary):

```
Hello
World
Hello
World
Hello
World
Hello
World
Hello
World
```

---

4. Write a Program for Demonstrate Thread.sleep()

```
public class ThreadExample4 {
    public static void main(String[] args) throws InterruptedException {
        for (int i = 1; i <= 3; i++) {
            System.out.println(i);
            Thread.sleep(500);
        }
    }
}
```

Output:

```
1
2
3
```

---

5. Write a Program for Thread.yield() usage

```
class YieldThread extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println(getName());
            Thread.yield();
        }
    }
}
public class ThreadExample5 {
    public static void main(String[] args) {
        new YieldThread().start();
        new YieldThread().start();
    }
}
```

Possible Output (order may vary):

```
Thread-0
Thread-1
Thread-0
Thread-1
Thread-0
Thread-1
```

---

6. Write a Program for Two threads: even and odd numbers

```
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 < 10; i += 2)
            System.out.println("Odd: " + i);
    }
}
public class ThreadExample6 {
    public static void main(String[] args) {
        new EvenThread().start();
        new OddThread().start();
    }
}
```

Possible Output (order may vary):

```
Even: 2
Odd: 1
Even: 4
Odd: 3
```

---

7. Write a Program for Three threads with different priorities

```
class PriorityThread extends Thread {
    public void run() {
        System.out.println(getName() + " Priority: " + getPriority());
    }
}
public class ThreadExample7 {
    public static void main(String[] args) {
        PriorityThread t1 = new PriorityThread();
        PriorityThread t2 = new PriorityThread();
        PriorityThread t3 = new PriorityThread();
        t1.setPriority(Thread.MIN_PRIORITY);
        t3.setPriority(Thread.MAX_PRIORITY);
        t1.start(); t2.start(); t3.start();
    }
}
```

Possible Output:

```
Thread-0 Priority: 1
Thread-1 Priority: 5
Thread-2 Priority: 10
```

---

8. Write a Program for Thread.join() usage

```
class JoinThread extends Thread {
    public void run() {
        for (int i = 1; i <= 3; i++)
            System.out.println(getName() + " " + i);
    }
}
public class ThreadExample8 {
    public static void main(String[] args) throws InterruptedException {
        JoinThread t1 = new JoinThread();
        t1.start();
        t1.join();
        System.out.println("Main thread finished after t1");
    }
}
```

Possible Output:

```
Thread-0 1
Thread-0 2
Thread-0 3
Main thread finished after t1
```

---

9. Write a Program for Stop a thread using a boolean flag

```
class StopThread extends Thread {
    volatile boolean running = true;
    public void run() {
        while (running) {
            System.out.println("Running...");
        }
    }
}
public class ThreadExample9 {
    public static void main(String[] args) throws InterruptedException {
        StopThread t = new StopThread();
        t.start();
        Thread.sleep(10);
        t.running = false;
    }
}
```

Possible Output:

```
Running...
Running...
```

---

10. Write a Program for Shared counter without synchronization (Race Condition)

```
class CounterThread extends Thread {
    static int counter = 0;
    public void run() {
        for (int i = 0; i < 1000; i++)
            counter++;
    }
}
```

```
public class ThreadExample10 {
    public static void main(String[] args) throws InterruptedException {
        CounterThread t1 = new CounterThread();
        CounterThread t2 = new CounterThread();
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Counter: " + CounterThread.counter);
    }
}
```

Possible Output:

```
Counter: 1827
```

---

11. Write a Program for Prevent race condition with synchronized

```
class SyncCounterThread extends Thread {
    static int counter = 0;
    public synchronized static void increment() {
        counter++;
    }
    public void run() {
        for (int i = 0; i < 1000; i++)
            increment();
    }
}
public class ThreadExample11 {
    public static void main(String[] args) throws InterruptedException {
        SyncCounterThread t1 = new SyncCounterThread();
        SyncCounterThread t2 = new SyncCounterThread();
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Counter: " + SyncCounterThread.counter);
    }
}
```

Output:

```
Counter: 2000
```

---

12. Write a Program for Synchronized block

```java
class SyncBlockExample extends Thread {
    static int counter = 0;
    public void run() {
        synchronized (SyncBlockExample.class) {
            for (int i = 0; i < 1000; i++)
                counter++;
        }
    }
}
public class ThreadExample12 {
    public static void main(String[] args) throws InterruptedException {
        SyncBlockExample t1 = new SyncBlockExample();
        SyncBlockExample t2 = new SyncBlockExample();
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Counter: " + SyncBlockExample.counter);
    }
}
```

Output:

```
Counter: 2000
```

---

13. Write a Program for BankAccount deposit/withdraw with synchronization

```java
class BankAccount {
    private int balance = 1000;
    public synchronized void deposit(int amt) { balance += amt; }
    public synchronized void withdraw(int amt) { balance -= amt; }
    public int getBalance() { return balance; }
}
```

```
public class ThreadExample13 {
    public static void main(String[] args) throws InterruptedException {
        BankAccount acc = new BankAccount();
        Thread t1 = new Thread(() -> { acc.deposit(500); });
        Thread t2 = new Thread(() -> { acc.withdraw(200); });
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Balance: " + acc.getBalance());
    }
}
```

Output:

```
Balance: 1300
```

---

14. Write a Program for Producer-Consumer with wait() and notify()

```
class Data {
    int value;
    boolean available = false;
    public synchronized void produce(int v) throws InterruptedException {
        while (available) wait();
        value = v;
        available = true;
        notify();
    }
    public synchronized int consume() throws InterruptedException {
        while (!available)
{wait();}
        available = false;
        notify();
        return value;
    }
}
public class ThreadExample14 {
    public static void main(String[] args) {
        Data data = new Data();
        Thread producer = new Thread(() -> {
            try { data.produce(10); } catch (Exception e) {}
        });
        Thread consumer = new Thread(() -> {
            try { System.out.println(data.consume()); } catch (Exception e) {}
        });
        producer.start(); consumer.start();
    }
}
```

Output:

```
10
```

---

15. Write a Program for One thread prints A–Z, another prints 1–26 alternately

```
class Printer {
    boolean letterTurn = true;
    public synchronized void printLetter(char c) throws InterruptedException {
        while (!letterTurn) wait();
        System.out.print(c + " ");
        letterTurn = false;
        notify();
    }
    public synchronized void printNumber(int n) throws InterruptedException {
        while (letterTurn)
{
wait();
}
        System.out.print(n + " ");
        letterTurn = true;
        notify();
    }
}
public class ThreadExample15 {
    public static void main(String[] args) {
        Printer p = new Printer();
        Thread t1 = new Thread(() -> {
            for (char c = 'A'; c <= 'Z'; c++) {
                try { p.printLetter(c); } catch (Exception e) {}
            }
        });
        Thread t2 = new Thread(() -> {
            for (int i = 1; i <= 26; i++) {
                try { p.printNumber(i); } catch (Exception e) {}
            }
        });
        t1.start(); t2.start();
    }
}
```

Possible Output:

```
A 1 B 2 C 3 ... Z 26
```

---

16. Write a Program for wait() and notifyAll()

```
class SharedData {
    public synchronized void task() throws InterruptedException {
        System.out.println(Thread.currentThread().getName() + " waiting");
        wait();
        System.out.println(Thread.currentThread().getName() + " resumed");
    }
    public synchronized void wakeUpAll() {
        notifyAll();
    }
}
public class ThreadExample16 {
    public static void main(String[] args) throws InterruptedException {
        SharedData sd = new SharedData();
        Thread t1 = new Thread(() -> { try { sd.task(); } catch (Exception e) {} });
        Thread t2 = new Thread(() -> { try { sd.task(); } catch (Exception e) {} });
        t1.start(); t2.start();
        Thread.sleep(500);
        sd.wakeUpAll();
    }
}
```

Possible Output:

```
Thread-0 waiting
Thread-1 waiting
Thread-0 resumed
Thread-1 resumed
```

---

17. Write a Program for Daemon thread prints time every second

```
import java.time.LocalTime;
public class ThreadExample17 {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> {
            while (true) {
                System.out.println(LocalTime.now());
                try { Thread.sleep(1000); } catch (Exception e) {}
            }
        });
        t.setDaemon(true);
        t.start();
        Thread.sleep(3000);
    }
}
```

Output (example):

```
12:30:15.123
```

```
12:30:16.123
12:30:17.123
```

---

18. Write a Program for Thread.isAlive()

```
public class ThreadExample18 {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> System.out.println("Running"));
        System.out.println(t.isAlive());
        t.start();
        Thread.sleep(100);
        System.out.println(t.isAlive());
    }
}
```

Possible Output:

```
false
Running
false
```

---

19. Write a Program for Thread group creation and management

```
public class ThreadExample19 {
    public static void main(String[] args) {
        ThreadGroup group = new ThreadGroup("MyGroup");
        Thread t1 = new Thread(group, () -> System.out.println("T1"));
        Thread t2 = new Thread(group, () -> System.out.println("T2"));
        t1.start(); t2.start();
        System.out.println("Active threads: " + group.activeCount());
    }
}
```

Possible Output:

```
Active threads: 2
T1
T2
```

20. Write a Program for Callable and Future

```
import java.util.concurrent.*;
public class ThreadExample20 {
    public static void main(String[] args) throws Exception {
        ExecutorService es = Executors.newSingleThreadExecutor();
        Callable<Integer> task = () -> 5 * 10;
        Future<Integer> future = es.submit(task);
        System.out.println("Result: " + future.get());
        es.shutdown();
    }
}
```

Output:

```
Result: 50
```