Wrapper Classes :

1. Check if character is digit ?

Program :

```
package Day7;
public class Digit {
 public static void main(String[] args) {
    char ch1 = '5';
    char ch2 = 'A';

    System.out.println("Is " + ch1 + " a digit? " + Character.isDigit(ch1));
    System.out.println("Is " + ch2 + " a digit? " + Character.isDigit(ch2));
 }
}
```

Output : Is 5 a digit? true

Is A a digit? false

2. Compare two strings ?

Program :

```
package Day7;
public class Strings {
 public static void main(String[] args) {
    String str1 = "Apple";
    String str2 = "Banana";
    int result = str1.compareTo(str2);
    if (result == 0) {
       System.out.println("Both strings are equal.");
    } else {
       System.out.println("Both strings are not equal");
    }
 }
}
```

Output : Both strings are not equal

3. Covert using valueOf method ?

Program :

```
package Day7;
public class Converting_str {
 public static void main(String[] args) {
    int num = 123;
    double price = 99.99;
    boolean status = true;

    String s1 = String.valueOf(num);
    String s2 = String.valueOf(price);
    String s3 = String.valueOf(status);

    System.out.println("Integer to String: " + s1);
    System.out.println("Double to String: " + s2);
    System.out.println("Boolean to String: " + s3);
 }
}
```

Output : Integer to String: 123

Double to String: 99.99

Boolean to String: true

4. Create Boolean wrapper usage ?

Program :

```
package Day7;
public class Bool {
 public static void main(String[] args) {
    Boolean bool1 = Boolean.valueOf(true);
    Boolean bool2 = Boolean.valueOf("false");
```

```
    System.out.println("bool1: " + bool1);
    System.out.println("bool2: " + bool2);

    System.out.println("Logical AND: " + (bool1 & bool2));
    System.out.println("Logical OR: " + (bool1 | bool2));
 }
}
```

Output : bool1: true

bool2: false

Logical AND: false

Logical OR: true

5. Convert null to wrapper classes ?

Program :

```
package Day7;
public class Null_conv {
 public static void main(String[] args) {
    String str = null;
    try {
       Integer num = Integer.valueOf(str);
       System.out.println("Converted number: " + num);
    } catch (Exception e) {
       System.out.println(e);
    }
 }
}
```

Output : java.lang.NumberFormatException: Cannot parse null string

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 ?

Program :

```
package Day7;
public class pass_by_Value {
    public static void main(String[] args) {
        int num = 10;
        System.out.println("Before : " + num);
        changeValue(num);
        System.out.println("After :  " + num);
    }
    static void changeValue(int n) {
        n = 20;
    }
}
```

Output : Before : 10

After : 10

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

Program :

```
package Day7;
public class Pass_By_value1 {
 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 swap: x = " + x + ", y = " + y);
     swap(x, y);
     System.out.println("After swap: x = " + x + ", y = " + y);
```

```
 }
}
```

Output : Before swap: x = 5, y = 10

After swap: x = 5, y = 10

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

Program :

```
package Day7;
public class Pass_By_Value_2 {
 public static void modify(int value) {
    value += 100;
    System.out.println("Inside method: value = " + value);
 }
 public static void main(String[] args) {
    int num = 20;
    System.out.println("Before method: num = " + num);
    modify(num);
    System.out.println("After method: num = " + num);
 }
}
```

Output : Before method: num = 20

Inside method: value = 120

After method: num = 20

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 ?

Program :

```
package Day7;
class Box {
 int length;
}
```

```
public class Pass_By_Ref {
 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);
 }
}
```

Output : Before: length = 10

After: length = 50

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?

Program :

```
package Day7;
class Person {
 String name;
 int age;
}
public class Pass_By_Ref_1 {
 public static void updatePerson(Person p) {
    p.name = "Dev";
    p.age = 30;
 }
```

```
 public static void main(String[] args) {
    Person person = new Person();
    person.name = "Muktha";
    person.age = 25;

    System.out.println("Before: " + person.name + ", " + person.age);
    updatePerson(person);
    System.out.println("After: " + person.name + ", " + person.age);
}
}
```

Output: Before: Muktha, 25

After: Dev, 30

6. 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 :

```
package Day7;
class Student {
 String name;
 int marks;
}
public class Student_Pass_By_Ref {
 public static void updateMarks(Student s, int newMarks) {
    s.marks = newMarks;
 }
 public static void main(String[] args) {
    Student stu = new Student();
    stu.name = "Dev";
    stu.marks = 75;

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

```
    updateMarks(stu, 90);
    System.out.println("After: " + stu.name + " - " + stu.marks);
 }
}
```

Output : Before: Dev - 75

After: Dev - 90

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

Program :

```
package Day7;
class MyObject {
 int value;
}
public class Test {
 public static void changeReference(MyObject obj) {
    obj = new MyObject();
    obj.value = 200;
 }
 public static void main(String[] args) {
    MyObject o = new MyObject();
    o.value = 100;

    System.out.println("Before: " + o.value);
    changeReference(o);
    System.out.println("After: " + o.value);
 }
}
```

Output : Before: 100

After: 100

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 ?

Program :

```
package Day7;

class Car {
 String model;
}
public class Test1 {
 public static void changeCar(Car c) {
    c = new Car();
    c.model = "Tesla";
 }

 public static void main(String[] args) {
    Car car = new Car();
    car.model = "BMW";

    System.out.println("Before: " + car.model);
    changeCar(car);
    System.out.println("After: " + car.model);
 }
}
```

Output : Before: BMW

After: BMW

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

Program :

```
package Day7;

class Data {
 int number;
}
```

```
public class Test2 {
 public static void change(int num) {
    num = 50;
 }
 public static void changeObj(Data d) {
    d.number = 50;
 }
 public static void main(String[] args) {
    int a = 10;
    Data obj = new Data();
    obj.number = 10;

    change(a);
    System.out.println("Primitive after change: " + a);

    changeObj(obj);
    System.out.println("Object after change: " + obj.number);
    }
}
```

Output : Primitive after change: 10

Object after change: 50

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

Program :

```
package Day7;
public class Test3 {
 public static void modify(int[] arr) {
    arr[0] = 100;
 }
```

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

Output : Before: 10

After: 100

Multithreading :

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

Program :

```
package Day7;
class MyThread1 extends Thread {
 public void run() {
    for (int i = 1; i <= 5; i++) {
       System.out.println(i);
    }
 }
}
public class Print {
 public static void main(String[] args) {
    MyThread1 t1 = new MyThread1();
    t1.start();
 }
}
```

Output : 1

2

3

4

5

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

Program :

```
package Day7;
class MyRunnable implements Runnable {
 public void run() {
    System.out.println("Current thread: " + Thread.currentThread().getName());
 }
}
public class Thread_Name {
 public static void main(String[] args) {
    Thread t = new Thread(new MyRunnable());
    t.start();
 }
}
```

Output : Current thread: Thread-0

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

Program :

```
package Day7;
class MessageThread extends Thread {
 String message;
 MessageThread(String message) {
    this.message = message;
 }
 public void run() {
    for (int i = 0; i < 5; i++) {
       System.out.println(message);
    }
 }
```

```
}

public class Print_Msg {
 public static void main(String[] args) {
    Thread t1 = new MessageThread("Hello from Thread 1");
    Thread t2 = new MessageThread("Hello from Thread 2");
    t1.start();
    t2.start();
 }
}
```

Output : Hello from Thread 2

Hello from Thread 2

Hello from Thread 2

Hello from Thread 2

Hello from Thread 2

Hello from Thread 1

Hello from Thread 1

Hello from Thread 1

Hello from Thread 1

Hello from Thread 1

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

Program :

```
package Day7;
public class Thread_Sleep {
 public static void main(String[] args) {
    for (int i = 1; i <= 3; i++) {
       System.out.println(i);
       try {
          Thread.sleep(1000); // 1 second pause
       } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }
 }
}
```

Output : 1

2

3

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

Program :

```
package Day7;
class MyThread extends Thread {
 public void run() {
    for (int i = 1; i <= 5; i++) {
        System.out.println(Thread.currentThread().getName() + " : " + i);
        Thread.yield();
    }
 }
}
public class Yield_Method {
 public static void main(String[] args) {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();
    t1.start();
    t2.start();
 }
}
```

Output : Thread-0 : 1

Thread-1 : 1

Thread-0 : 2

Thread-1 : 2

Thread-0 : 3

Thread-1 : 3

Thread-0 : 4

Thread-1 : 4

Thread-1 : 5

Thread-0 : 5

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

Program :

```
package Day7;
class MyThread1 extends Thread {
 public void run() {
    for (int i = 2; i <= 10; i += 2) {
        System.out.println("Even: " + i);
    }
 }
}
class MyThread2 extends Thread {
 public void run() {
    for (int i = 1; i < 10; i += 2) {
        System.out.println("Odd: " + i);
    }
 }
}
public class EvenOdd {
 public static void main(String[] args) {
    new MyThread1().start();
    new MyThread2().start();
 }
}
```

Output : Even: 2

Even: 4

Even: 6

Even: 8

Even: 10

Odd: 1

Odd: 3

Odd: 5

Odd: 7

Odd: 9

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

Program :

```
package Day7;
class PriorityThread extends Thread {
 public void run() {
    System.out.println(getName() + " Priority: " + getPriority());
 }
}

public class Multi_thread {
 public static void main(String[] args) {
    PriorityThread t1 = new PriorityThread();
    PriorityThread t2 = new PriorityThread();
    PriorityThread t3 = new PriorityThread();

    t1.setPriority(Thread.MIN_PRIORITY);
    t2.setPriority(Thread.NORM_PRIORITY);
    t3.setPriority(Thread.MAX_PRIORITY);

    t1.start();
```

```
    t2.start();
    t3.start();
 }
}
```

Output : Thread-1 Priority: 5

Thread-2 Priority: 10

Thread-0 Priority: 1

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

Program :

```
package Day7;
class MyThread extends Thread {
 public void run() {
    for (int i = 1; i <= 3; i++) {
        System.out.println(getName() + ": " + i);
    }
 }
}

public class Thread_join {
 public static void main(String[] args) throws InterruptedException {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();
    t1.start();
    t1.join();
    t2.start();
 }
}
```

Output :

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

Program :

```
package Day7;
class MyThread extends Thread {
 volatile boolean running = true;
 public void run() {
    while (running) {
       System.out.println("Thread is running...");
    }
 }
}
public class Bool_flag {
 public static void main(String[] args) throws InterruptedException {
    MyThread t = new MyThread();
    t.start();
    Thread.sleep(1000);
    t.running = false;
 }
}
```

Output : Thread is running...

```
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
Thread is running...
```

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

Thread is running...

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

Program :

```
package Day7;
class Counter {
	int count = 0;
	void increment() {
		count++;
	}
}
public class Test {
	public static void main(String[] args) {
		Counter counter = new Counter();
		Runnable task = () -> {
			for (int i = 0; i < 1000; i++) {
				counter.increment();
			}
		};
		Thread t1 = new Thread(task);
		Thread t2 = new Thread(task);
```

```
		t1.start();
		t2.start();
		try {
			t1.join();
			t2.join();
		} catch (Exception e) {
		}
		System.out.println("Final Count: " + counter.count);
	}
}
```

Output : Final Count: 2000

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

Program :

```
package Day7;
class Count {
	int count = 0;
	synchronized void increment() {
		count++;
	}
}
public class Test12 {
	public static void main(String[] args) {
		Count counter = new Count();
		Runnable task = () -> {
			for (int i = 0; i < 1000; i++) {
				counter.increment();
			}
		};
		Thread t1 = new Thread(task);
		Thread t2 = new Thread(task);
```

```
		t1.start();
		t2.start();
		try {
			t1.join();
			t2.join();
		} catch (Exception e) {
		}
		System.out.println("Final Count: " + counter.count);
	}
}
```

Output : Final Count: 2000

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

Program :

```
package Day7;
class Counter {
	int count = 0;
	void increment() {
		synchronized (this) {
			count++;
		}
	}
}
public class Test2 {
	public static void main(String[] args) {
		Counter counter = new Counter();
		Runnable task = () -> {
			for (int i = 0; i < 1000; i++) {
				counter.increment();
			}
		};
```

```
                Thread t1 = new Thread(task);
                Thread t2 = new Thread(task);
                t1.start();
                t2.start();
                try {
                        t1.join();
                        t2.join();
                } catch (Exception e) {
                }
                System.out.println("Final Count: " + counter.count);
        }
}
```

Output : Final Count: 2000

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

Program :

```
Package Day7;
class BankAccount {
   private int balance = 1000;
   public synchronized void deposit(int amount) {
      balance += amount;
      System.out.println(Thread.currentThread().getName() + " deposited " + amount + ", New Balance: " + balance);
   }
   public synchronized void withdraw(int amount) {
      if (balance >= amount) {
         balance -= amount;
         System.out.println(Thread.currentThread().getName() + " withdraw " + amount + ", Remaining Balance: " + balance);
      } else {
```

```
            System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + " but insufficient funds. Balance: " + balance);
        }
    }
    public int getBalance() {
        return balance;
    }
}

class Bank implements Runnable {
    private BankAccount account;

    Bank(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.deposit(275);
        account.withdraw(125);
    }

public class Bank1 {
    public static void main(String[] args) {
        BankAccount b = new BankAccount();

        Thread t1 = new Thread(new Bank(b));
        Thread t2 = new Thread(new Bank(b));
        t1.start();
        t2.start();
    }
}
```

Output : Thread-0 deposited 275, New Balance: 1275

Thread-0 withdraw 125, Remaining Balance: 1150

Thread-1 deposited 275, New Balance: 1425

Thread-1 withdraw 125, Remaining Balance: 1300

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

Program :

```
package MultiThreading;
class Shared {
	int num;
	boolean ready = false;

	synchronized void produce(int n) {
		if(ready) return;
		num = n;
		System.out.println("Produced : " + num);
		ready = true;
		notify();
	}
	synchronized void consume() {
		while(!ready) {
			try {
				wait();
			} catch (Exception e ) {}
		}
		System.out.println("Consumed : " + num);
	}
}
public class WaitDemo {

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

```
                Shared s = new Shared();

                new Thread(() -> s.produce(6)).start();
                new Thread(() -> s.consume()).start();
        }

}
```

Output : Produced : 6

Consumed : 6

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

Program :

```
package Day7;
class PrintData {
        boolean letterTurn = true;
}
public class PrintA_Z {
        public static void main(String[] args) {
                PrintData data = new PrintData();

                Thread t1 = new Thread(() -> {
                        for (char c = 'A'; c <= 'Z'; c++) {
                                synchronized (data) {
                                        while (!data.letterTurn) {
                                                try {
                                                        data.wait();
                                                } catch (Exception e) {
                                                }
                                        }
                                        System.out.print(c + " ");
                                        data.letterTurn = false;
```

```
                        data.notify();
                    }
                }
            });

            Thread t2 = new Thread(() -> {
                for (int i = 1; i <= 26; i++) {
                    synchronized (data) {
                        while (data.letterTurn) {
                            try {
                                data.wait();
                            } catch (Exception e) {
                            }
                        }
                        System.out.print(i + " ");
                        data.letterTurn = true;
                        data.notify();
                    }
                }
            });

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

Output : A 1 B 2 C 3 D 4 E 5 F 6 G 7 H 8 I 9 J 10 K 11 L 12 M 13 N 14 O 15 P 16 Q 17 R 18 S 19 T 20 U 21 V 22 W 23 X 24 Y 25 Z 26

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

Program :

```
package Day7;
```

```
class SharedResource {
 synchronized void printMessage(String msg) {
    System.out.println(msg);
    notifyAll();
 }
}

public class Thread_Comm {
 public static void main(String[] args) {
    SharedResource res = new SharedResource();

    Runnable task = () -> {
       synchronized (res) {
          try { res.wait(); } catch (Exception e) {}
          res.printMessage("Thread " + Thread.currentThread().getName() );
       }
    };

    new Thread(task, "T1").start();
    new Thread(task, "T2").start();
    new Thread(task, "T3").start();

    try { Thread.sleep(1000); } catch (Exception e) {}
    synchronized (res) {
       res.printMessage("Main thread notifying all...");
    }
 }
}
```

Output : Main thread notifying all...

Thread T2

Thread T1

Thread T3

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

Program :

```
package Day7;
import java.time.LocalTime;
public class Print_Time {
	public static void main(String[] args) {
		Thread daemon = new Thread(() -> {
			while (true) {
				System.out.println("Time: " + LocalTime.now());
				try {
					Thread.sleep(1000);
				} catch (Exception e) {
				}
			}
		});
		daemon.setDaemon(true);
		daemon.start();

		try {
			Thread.sleep(5000);
		} catch (Exception e) {
		}
		System.out.println("Main thread finished.");
	}
}
```

Output : Time: 09:10:07.453113900

Time: 09:10:08.465335600

Time: 09:10:09.478378400

Time: 09:10:10.493068100

Time: 09:10:11.507847300

Main thread finished.

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

Program :

```
package Day7;
public class isALive {
 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());
 }
}
```

Output : Before start: false

Thread running...

After start: true

After join: false

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

Program :

```
package Day7;
public class Thread_group {
 public static void main(String[] args) {
    ThreadGroup group = new ThreadGroup("MyGroup");
```

```
    Thread t1 = new Thread(group, () -> System.out.println("Thread 1 running"));
    Thread t2 = new Thread(group, () -> System.out.println("Thread 2 running"));

    t1.start();
    t2.start();
    System.out.println("Active Threads: " + group.activeCount());
 }
}
```

Output : Thread 2 running

Thread 1 running

Active Threads: 2

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

Program :

```
package Day7;
import java.util.concurrent.*;
public class Callable_Mul {
        public static void main(String[] args) throws Exception {
                ExecutorService executor = Executors.newSingleThreadExecutor();

                Callable<Integer> task = () -> {
                        int result = 5 * 10;
                        return result;
                };

                Future<Integer> future = executor.submit(task);
                System.out.println("Result: " + future.get());

                executor.shutdown();
        }
```

}

Output : Result: 50