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.**

Program:

```
public class WrapperClassExamples {
    public static void main(String[] args) {
        char ch = '9';
        boolean isDigit = Character.isDigit(ch);
        System.out.println(ch + " is digit? " + isDigit);
        String str1 = "hello";
        String str2 = "Hello";
        boolean isEqual = str1.equals(str2);
        System.out.println("Strings equal? " + isEqual);
        int num = 779;
        String numSt = String.valueOf(num);
        System.out.println("String from int: " + numSt);
        double d = 88.67;
        Double dObj = Double.valueOf(d);
        System.out.println("Double object: " + dObj);
        Boolean boolObj = Boolean.valueOf(true);
        System.out.println("Boolean wrapper: " + boolObj);
        String nullSt = null;
        Integer intObj = nullStr == null ? null : Integer.valueOf(nullSt);
```

```
        System.out.println("Integer object from null string: " + intObj);
    }
}
```

Output:

9 is digit? true

Strings equal? false

String from int: 779

Double object: 88.67

Boolean wrapper: true

Integer object from null string: null

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:

```
public class PassByValueDemo {
    static void changeValue(int num) {
        System.out.println("Inside method, before change: " + num);
        num = 100;
        System.out.println("Inside method, after change: " + num);
    }
    public static void main(String[] args) {
        int original = 50;
        System.out.println("Before method call: " + original);
```

```
        changeValue(original);
        System.out.println("After method call: " + original);
    }
}
```

Output:

Before method call: 20

Inside method, before change: 20

Inside method, after change: 10

After method call: 20

---

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

Ans)

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

Output:

Before method call: x = 10, y = 20

Inside method before swap: a = 10, b = 20

Inside method after swap: a = 20, b = 10

After method call: x = 10, y = 20

---

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:

```
public class PrimitivePassDemo {
    static void modifyPrimitives(int a, double b, boolean c) {
        System.out.println(before: a = " + a + ", b = " + b + ", c = " + c);
        a = 10;
        b = 20.5;
        c = false;
        System.out.println("after : a = " + a + ", b = " + b + ", c = " + c);
    }
    public static void main(String[] args) {
        int x = 10;
        double y = 26.2;
        boolean z = true;
```

```
        System.out.println("Before method call: x = " + x + ", y = " + y + ", z = " + z);
        modifyPrimitives(x, y, z);
        System.out.println("After method call: x = " + x + ", y = " + y + ", z = " + z);
    }
}
```

Output:

Before method call: x = 10, y = 26.2, z = true

before: a = 10, b = 20.5, c = true

after: a = 10, b = 20.5, c = false

After method call: x = 10, y = 20.5, z = true

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.

Program

```
class Box {
    int length;
    Box(int length) {
        this.length = length;
    }
}
public class CallByReferenceDemo {
    static void modifyLength(Box box) {
        System.out.println("before: length = " + box.length);
```

```
        box.length = 70;
        System.out.println("after: length = " + box.length);
    }
    public static void main(String[] args) {
        Box myBox = new Box(30);
        System.out.println("Before method call: length = " + myBox.length);
        modifyLength(myBox);
        System.out.println("After method call: length = " + myBox.length);
    }
}
```

Output:

Before method call: length = 10

before: length = 10

after: length = 50

After method call: 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:

```
    class Dresses {
    String model;
    Dresses(String model) {
        this.model = model;
    }
}
```

```
public class ModifyObjectFields {
   static void changeModel(Dresses dress) {
      dress.model = "shirts";
   }
   public static void main(String[] args) {
      Dresses myDress = new Dress("Formal dress");
      System.out.println("Before: " + myDress.model);
      changeModel(myDress);
      System.out.println("After: " + myDress.model);
   }
}
```

Output:

Before: Shirts

After: Formal Dress

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:

```
    class Student {
       String name;
       int marks;
       Student(String name, int marks) {
          this.name = name;
          this.marks = marks;
       }
```

```
    void updateMarks(int newMarks) {
        marks = newMarks;
    }
}
public class UpdateStudentMarks {
    public static void main(String[] args) {
        Student s = new Student("Nikhitha", 75);
        System.out.println("Before update: " + s.marks);
        s.updateMarks(90);
        System.out.println("After update: " + s.marks);
    }
}
```

Output:

```
Before update: 75
After update: 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:

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
}
public class CallByValue {
```

```
    static void changeReference(Person p) {
        p = new Person("Srihari");
    }
    public static void main(String[] args) {
        Person person = new Person("Jayanth");
        System.out.println("Before: " + person.name);
        changeReference(person);
        System.out.println("After: " + person.name);
    }
}
```

Output:

Before: Jayanth

After: Jayanth

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. give solutions for these questions in simple way seperatly

Program:

```
class Box {
    int size;
    Box(int size) {
        this.size = size;
    }
}
public class NewobjAssign {
    static void assignNewBox(Box box) {
        box = new Box(100);  // New object assigned locally
```

```
    }
    public static void main(String[] args) {
        Box myBox = new Box(10);
        System.out.println("Before: " + myBox.size);
        assignNewBox(myBox);
        System.out.println("After: " + myBox.size);
    }
}
```

Output:

Before: 10

After: 10

---

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

Primitive data types: Primitive data types are the basic types like int, float, char, etc. Values are stored directly and are already predefined in Java.

Non-primitive data types: Non-primitive data types include arrays, classes, interfaces, and Strings. They can store multiple values and you can call methods on them. Also, they are created by the programmer or Java libraries, not predefined like primitive data types.

Example for Primitive and Non primitive data type

```
public class PrimitiveDemo {
 public static void main(String[] args) {
    byte b = 10;
    short s = 1000;
    int i = 100;
    long l = 100000000L;
```

```
        float f = 8.89f;
        double d = 55.99;
        char c = 'A';
        boolean bole = true;

        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
        System.out.println("char: " + c);
        System.out.println("boolean: " + bole);
    }
}
```

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

Program:

```
public class SimulateCallByReference {
    static void changeValue(int[] arr) {
        arr[0] = 100;
    }
    public static void main(String[] args) {
        int[] values = {10};
        System.out.println("Before: " + values[0]);
        changeValue(values);
        System.out.println("After: " + values[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

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

Output:

1

2

3

4

5

---

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

Program

```
class RunnableDemo implements Runnable {
   public void run() {
      System.out.println("Thread name: " + Thread.currentThread().getName());
   }
}
public class Demo2 {
   public static void main(String[] args) {
      Thread t = new Thread(new RunnableDemo());
      t.start();
   }
}
```

Output:

Thread name: Thread-0

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

Program

```
class Message implements Runnable {
   private String msg;
   Message(String msg) { this.msg = msg; }
```

```
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(msg);
        }
    }
}
public class creating_threads {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Message("Hello"));
        Thread t2 = new Thread(new Message("World"));
        t1.start();
        t2.start();
    }
}
```

Output:

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

4. Demonstrate Thread.sleep() between numbers 1 to 3.

Program:

```
public class thread_exam {
   public static void main(String[] args) throws InterruptedException {
      for (int i = 1; i <= 3; i++) {
         System.out.println(i);
         Thread.sleep(1000);
      }
   }
}
```

Output:

1

(wait 1 second)

2

(wait 1 second)

3

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

Program:

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

```
        }
    }
}
public class Thread_demo {
    public static void main(String[] args) {
        YieldExam t1 = new YieldExam();
        YieldExam t2 = new YieldExam();
        t1.start();
        t2.start();
    }
}
```

Output:

```
Thread-0: 1
Thread-1: 1
Thread-0: 2
Thread-1: 2
```

---

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

Program:

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

```
    }
}
class Odd_Thread extends Thread {
    public void run() {
        for (int i = 1; i <= 9; i += 2) {
            System.out.println("Odd: " + i);
        }
    }
}
public class EvenOddExam {
    public static void main(String[] args) {
        new Even_Thread().start();
        new Odd_Thread().start();
    }
}
```

Output:

```
Odd: 1
Even: 2
Odd: 3
Even: 4
Odd: 5
Even: 6
Odd: 7
Even: 8
Odd: 9
```

Even: 10

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

Program:

```
class Priority_Thread extends Thread {
   Priority_Thread(String name) {
      super(name);
   }
   public void run() {
      for (int i = 1; i <= 3; i++) {
         System.out.println(getName() + " running");
      }
   }
}
public class PriorityExam {
   public static void main(String[] args) {
      Priority_Thread t1 = new Priority_Thread("Medium");
      Priority_Thread t3 = new Priority_Thread("High");
      t1.setPriority(Thread.MIN_PRIORITY);
      t2.setPriority(Thread.NORM_PRIORITY);
      t3.setPriority(Thread.MAX_PRIORITY);
      t1.start();
      t2.start();
      t3.start();
```

```
    }
}
```

Output:

```
High running
High running
High running
Medium running
Medium running
Medium running
Low running
Low running
Low running
```

---

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

Program:

```
class Join_Thread extends Thread {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(" thread: " + i);
        }
    }
}
public class Join_Exam{
    public static void main(String[] args) throws InterruptedException {
```

```
        Join_Thread t = new Join_Thread();
        t.start();
        t.join(); // Wait until child finishes
        System.out.println("Main thread ");
    }
}
```

Output:

Child thread: 1

Child thread: 2

Child thread: 3

Main thread continues after child finishes

---

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

Program:

```
class Stop_Thread extends Thread {
    volatile boolean running = true;
    public void run() {
        int i = 1;
        while (running) {
            System.out.println("Count: " + i++);
        }
    }
}
public class StopFlagExam {
    public static void main(String[] args) throws InterruptedException {
```

```
        Stop_Thread t = new Stop_Thread();
        t.start();
        Thread.sleep(50);
        t.running = false;
    }
}
```

Output:

Count: 1

Count: 2

Count: 3

Count: 4

...

---

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

Program

```
class Counter {
    int count = 0;
    void increment() {
        count++;
    }
}
class Counter_Thread extends Thread {
    Counter c;
    Counter_Thread(Counter c) { this.c = c; }
```

```
    public void run() {
        for (int i = 0; i < 1000; i++) {
            c.increment();
        }
    }
}
public class Race_Example {
    public static void main(String[] args) throws InterruptedException {
        Counter c = new Counter();
        Counter_Thread t1 = new Counter_Thread(c);
        Counter_Thread t2 = new Counter_Thread(c);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Final count: " + c.count);
    }
}
```

Output:

Final count: 1765

---

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

Program:

```
class Counter_exam {
    private int count = 0;
    public synchronized void increment() {
        count++;
    }
    public int getCount() {
        return count;
    }
}
public class Sync_Example {
    public static void main(String[] args) throws InterruptedException {
        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();
        t1.join();
        t2.join();
        System.out.println("Final Count: " + counter.getCount());
    }
```

```
}
```

Output:

Final Count: 2000

---

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

Program

```
class Sync_Example {
   private int count = 0;
   public void increment() {
      synchronized (this) {
         count++;
      }
   }
   public int getCount() {
      return count;
   }
}
public class Sync_Demo {
   public static void main(String[] args) throws InterruptedException {
      SyncBlockExample obj = new SyncBlockExample();
      Runnable task = () -> {
         for (int i = 0; i < 1000; i++) {
            obj.increment();
         }
```

```
        };
        Thread t1 = new Thread(task);
        Thread t2 = new Thread(task);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Count: " + obj.getCount());
    }
}
```

Output:

Count: 2000

---

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

Program

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

```
            System.out.println(Thread.currentThread().getName() + " withdrew " +
amount + ". Balance: " + balance);
        } else {
            System.out.println(Thread.currentThread().getName() + " tried to
withdraw " + amount + " but insufficient funds.");
        }
    }
}
public class BankAccount_exam {
    public static void main(String[] args) {
        Bank_Account account = new Bank_Account();
        Runnable depositTask = () -> account.deposit(500);
        Runnable withdrawTask = () -> account.withdraw(700);
        Thread t1 = new Thread(depositTask, "Thread-1");
        Thread t2 = new Thread(withdrawTask, "Thread-2");
        t1.start();
        t2.start();
    }
}
```

Output:

Thread-1 deposited 500. Balance: 1500

Thread-2 withdrew 700. Balance: 800

---

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

Program:

```
class Demo {
```

```
    private int data;
    private boolean hasData = false;
    public synchronized void produce(int value) throws InterruptedException {
        while (hasData) {
            wait();
        }
        data = value;
        System.out.println("Produced: " + data);
        hasData = true;
        notify();
    }
    public synchronized void consume() throws InterruptedException {
        while (!hasData) {
            wait();
        }
        System.out.println("Consumed: " + data);
        hasData = false;
        notify();
    }
}
public class Producer_Consumer {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();
        Thread producer = new Thread(() -> {
            for (int i = 1; i <= 5; i++) {
```

```
            try {
                resource.produce(i);
            } catch (InterruptedException e) {}
        }
    });
    Thread consumer = new Thread(() -> {
        for (int i = 1; i <= 5; i++) {
            try {
                resource.consume();
            } catch (InterruptedException e) {}
        }
    });
    producer.start();
    consumer.start();
  }
}
```

Output:

```
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```

Produced: 5

Consumed: 5

---

15. Create a program where one thread prints A-Z and another prints 1-26 alternately.give solutions with outputs also

Program:

```
class Printer {
   boolean letter = true;
   public synchronized void printLetter(char letter) throws InterruptedException
{
      while (!letter) {
         wait();
      }
      System.out.print(letter + " ");
      letter = false;
      notify();
   }
   public synchronized void printNumber(int num) throws InterruptedException
{
      while (letter) {
         wait();
      }
      System.out.print(num + " ");
      letter = true;
      notify();
   }
```

```
}
public class AlternatePrint {
    public static void main(String[] args) {
        Printer printer = new Printer();
        Thread t1 = new Thread(() -> {
            for (char c = 'A'; c <= 'Z'; c++) {
                try {
                    printer.printLetter(c);
                } catch (InterruptedException e) {}
            }
        });
        Thread t2 = new Thread(() -> {
            for (int i = 1; i <= 26; i++) {
                try {
                    printer.printNumber(i);
                } catch (InterruptedException e) {}
            }
        });
        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:

```
class Message {
   private String msg;
   public synchronized void writeMessage(String m) {
      msg = m;
      notifyAll();
   }
   public synchronized String readMessage() {
      try { wait(); } catch (InterruptedException e) {}
      return msg;
   }
}
public class InterThreadCommunication {
   public static void main(String[] args) {
      Message m = new Message();
      Thread writer = new Thread(() -> m.writeMessage("Hello from Writer"));
      Thread reader = new Thread(() -> System.out.println("Reader got: " +
m.readMessage()));
      reader.start();
      writer.start();
   }
}
```

Output:

Reader got: Hello from Writer

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

Program

```
import java.time.LocalTime;
public class DaemonThread {
   public static void main(String[] args) {
      Thread daemon = new Thread(() -> {
         while (true) {
            System.out.println("Time: " + LocalTime.now());
            try {
              Thread.sleep(1000);
             }
           catch (InterruptedException e) {}
         }
      });
      daemon.setDaemon(true);
      daemon.start();
      try
   {
  Thread.sleep(3500);
   }
  catch (InterruptedException e) {}
      System.out.println("Main thread finished");
   }
}
```

Output:

Time: 10:15:20.123

Time: 10:15:21.124

Time: 10:15:22.124

Time: 10:15:23.124

Main thread finished

---

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

Program

```
public class IsAliveExam {
   public static void main(String[] args) throws InterruptedException {
      Thread t = new Thread(() -> {
         for (int i = 1; i <= 3; i++) {
            System.out.println("Thread running: " + i);
            try { Thread.sleep(500); } catch (InterruptedException e) {}
         }
      });
      System.out.println("Before start: " + t.isAlive());
      t.start();
      System.out.println("After start: " + t.isAlive());
      t.join();
      System.out.println("After finish: " + t.isAlive());
   }
}
```

Output:

Before start: false

After start: true

Thread running: 1

Thread running: 2

Thread running: 3

After finish: false

---

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

Program

```
public class ThreadGroupExam {
   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 in group: " + group.activeCount());
   }
}
```

Output:

Thread 1 running

Thread 2 running

Active threads in group: 2

20. Create a thread that performs a simple task (like multiplication) and returns result using Callable and Future. give simple codes for these questions .give solutions and also give outputs for all.give solutions and give output.

Program:

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

public class CallableExam {

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

      ExecutorService executor = Executors.newSingleThreadExecutor();

      Callable<Integer> task = () -> 5 * 10;

      Future<Integer> result = executor.submit(task);

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

      executor.shutdown();

   }

}
```

Output:

```
Result: 50
```