Wrapper classes

1. Check if character is a Digit

```
public class checkdigit {
    public static void main(String[] args) {
        char ch = '7';
        if (Character.isDigit(ch)) {
            System.out.println(ch + " is a digit.");
        } else {
            System.out.println(ch + " is NOT a digit.");
        }
    }
}

Output: 7 is a digit.
```

2. Compare two Strings

```
public class comparestrings {
    public static void main(String[] args) {
        String str1 = "hello";
        String str2 = "hello";
        String str3 = new String("hello");
        if (str1 == str2) {
            System.out.println("str1 == str2 : They are the same object");
        } else {
            System.out.println("str1 == str2 : They are different objects");
        }
        if (str1 == str3) {
            System.out.println("str1 == str3 : They are the same object");
        } else {
            System.out.println("str1 == str3 : They are different objects");
        }
        if (str1.equals(str3)) {
            System.out.println("str1.equals(str3) : Their contents are the same");
        } else {
            System.out.println("str1.equals(str3) : Their contents are
different");
        }
    }
}


Output: str1 == str2 : They are the same object

str1 == str3 : They are different objects

str1.equals(str3) : Their contents are the same
```

3. Convert using valueof method

```
public class valueofexample {
    public static void main(String[] args) {
        int num = 100;
        String str = String.valueOf(num);
        System.out.println("String value: " + str);
        int num2 = Integer.parseInt(str);
        System.out.println("Integer value: " + num2);
    }
}


Output: String value: 100

Integer value: 100
```

4. Create Boolean Wrapper usage

```
public class booleanwrapper {
    public static void main(String[] args) {
        Boolean boolObj1 = Boolean.valueOf(true);
        Boolean boolObj2 = Boolean.FALSE;
        System.out.println("boolObj1 = " + boolObj1);
        System.out.println("boolObj2 = " + boolObj2);
        boolean primBool = boolObj1.booleanValue();
        System.out.println("Primitive boolean = " + primBool);
    }
}

Output: boolObj1 = true

boolObj2 = false

Primitive boolean = true
```

5. Convert null to wrapper classes

```
public class nulltowrapper {

    public static void main(String[] args) {

        Integer num = null;



        try {

            int primitiveNum = num;
```

```
            System.out.println("Value: " + primitiveNum);

        } catch (NullPointerException e) {

            System.out.println("Cannot convert null wrapper to primitive!");

        }

    }

}

Output: Cannot convert null wrapper to primitive!
```

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

```
public class swapdemo {
    public static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
        System.out.println("Inside swap method: a = " + a + ", b = " + b);
    }
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        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 = 10, y = 20

Inside swap method: a = 20, b = 10

After swap: x = 10, y = 20
```

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

```
public class primitivepass {
    public static void changeValues(int a, double b, boolean c) {
        a = 100;
        b = 20.5;
        c = false;
        System.out.println("Inside method: a = " + a + ", b = " + b + ", c = " +
c);
    }
    public static void main(String[] args) {
        int x = 10;
        double y = 5.5;
        boolean z = true;
        System.out.println("Before method call: x = " + x + ", y = " + y + ", z =
" + z);
        changeValues(x, y, z);
        System.out.println("After method call: x = " + x + ", y = " + y + ", z = "
+ z);
    }
}

Output: Before method call: x = 10, y = 5.5, z = true

Inside method: a = 100, b = 20.5, c = false

After method call: x = 10, y = 5.5, 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.

```
class Box {
    int length;
    Box(int length) {
        this.length = length;
    }
}
```

```
public class boxdemo {
    public static void changeLength(Box box) {
        box.length = 100;
        System.out.println("Inside method, length = " + box.length);
    }
    public static void main(String[] args) {
        Box myBox = new Box(50);
        System.out.println("Before method call, length = " + myBox.length);
        changeLength(myBox);
        System.out.println("After method call, length = " + myBox.length);
    }
}

Output: Before method call, length = 50

Inside method, length = 100

After method call, length = 100
```

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.

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
}
public class objdemo {
    public static void changeName(Person p) {
        p.name = "Alice";
        System.out.println("Inside method: name = " + p.name);
    }
    public static void main(String[] args) {
        Person person = new Person("Bob");
        System.out.println("Before method call: name = " + person.name);
        changeName(person);
        System.out.println("After method call: name = " + person.name);
    }
}

Output: Before method call: name = Bob

Inside method: name = Alice

After method call: name = Alice
```

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.

```
class Student {
    String name;
    int marks;
    Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
}
public class stud_demo {
    public static void updateMarks(Student student, int newMarks) {
        student.marks = newMarks;  // Update marks
        System.out.println("Inside method: marks = " + student.marks);
    }
    public static void main(String[] args) {
        Student s = new Student("Ravi", 75);
        System.out.println("Before update: " + s.name + " has marks " + s.marks);
        updateMarks(s, 90);
        System.out.println("After update: " + s.name + " has marks " + s.marks);
    }
}

Output: Before update: Ravi has marks 75

Inside method: marks = 90

After update: Ravi has marks 90
```

---

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

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
}
public class callbyvaluedemo {
    public static void changeReference(Person p) {
        p = new Person("Charlie"); // Reassign p to a new object
        System.out.println("Inside method: p.name = " + p.name);
    }
    public static void main(String[] args) {
        Person person = new Person("Bob");
        System.out.println("Before method call: person.name = " + person.name);
        changeReference(person);
```

```
        System.out.println("After method call: person.name = " + person.name);
    }
}

Output: Before method call: person.name = Bob

Inside method: p.name = Charlie

After method call: person.name = Bob
```

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.

```
class Car {
    String model;
    Car(String model) {
        this.model = model;
    }
}
public class referencedemo {
    public static void assignNewObject(Car car) {
        car = new Car("Tesla");
        System.out.println("Inside method: car.model = " + car.model);
    }
    public static void main(String[] args) {
        Car myCar = new Car("Toyota");
        System.out.println("Before method call: myCar.model = " + myCar.model);
        assignNewObject(myCar);
        System.out.println("After method call: myCar.model = " + myCar.model);
    }
}

Output: Before method call: myCar.model = Toyota

Inside method: car.model = Tesla

After method call: myCar.model = Toyota
```

---

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

```
package assignment7;
```

```
public class primitivedemo {
    public static void changeValue(int num) {
        num = 100;
    }
    public static void main(String[] args) {
        int x = 50;
        changeValue(x);
        System.out.println("After method call, x = " + x);
    }
}
Output: After method call, x = 50
Non-primitive
package assignment7;
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
}
public class nonprimitive {
    public static void changeName(Person p) {
        p.name = "Alice";
    }
    public static void main(String[] args) {
        Person person = new Person("Bob");
        changeName(person);
        System.out.println("After method call, name = " + person.name);
    }
}
Output: After method call, name = Alice
```

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

Yes! In Java, you **can simulate call by reference** behavior by using a **wrapper class** or an **array**, because objects and arrays are passed by reference (well, technically the reference is passed by value, but you can modify the object's content).

```
class IntWrapper {
    int value;
    IntWrapper(int value) {
        this.value = value;
    }
}
public class callbyreference {
    public static void changeWithWrapper(IntWrapper num) {
        num.value = 100;
        System.out.println("Inside changeWithWrapper: value = " + num.value);
    }
    public static void changeWithArray(int[] arr) {
        arr[0] = 200;
        System.out.println("Inside changeWithArray: arr[0] = " + arr[0]);
```

```
    }
    public static void main(String[] args) {
        IntWrapper myNum = new IntWrapper(50);
        int[] numbers = {50};
        System.out.println("Before method calls:");
        System.out.println("Wrapper value = " + myNum.value);
        System.out.println("Array value = " + numbers[0]);
        changeWithWrapper(myNum);
        changeWithArray(numbers);
        System.out.println("After method calls:");
        System.out.println("Wrapper value = " + myNum.value);
        System.out.println("Array value = " + numbers[0]);
    }
}

Output: Before method calls:

Wrapper value = 50

Array value = 50

Inside changeWithWrapper: value = 100

Inside changeWithArray: arr[0] = 200

After method calls:

Wrapper value = 100

Array value = 200
```

MultiThreading

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

```
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class thread_demo {
    public static void main(String[] args) {
        MyThread t = new MyThread();
```

```
        t.start();
    }
}

Output: 1

2

3

4

5
```

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

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

Output: Thread running: Thread-0
```

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

```
class MessagePrinter implements Runnable {
    private String message;
    MessagePrinter(String message) {
        this.message = message;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(message + " - " + i);
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
}
public class twothreads {
    public static void main(String[] args) {
        Thread thread1 = new Thread(new MessagePrinter("Hello from Thread 1"));
        Thread thread2 = new Thread(new MessagePrinter("Hello from Thread 2"));

        thread1.start();
        thread2.start();
    }
}
Output: Hello from Thread 1 - 1

Hello from Thread 2 - 1

Hello from Thread 2 - 2

Hello from Thread 1 - 2

Hello from Thread 2 - 3

Hello from Thread 1 - 3

Hello from Thread 1 - 4

Hello from Thread 2 - 4

Hello from Thread 1 - 5

Hello from Thread 2 - 5
```

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

```
public class sleepdemo {
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            System.out.println(i);
            try {
                Thread.sleep(1000);  // Pause for 1 second (1000 milliseconds)
            } 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.

```
class MyThread1 extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(getName() + " - " + i);
            if (i == 3) {
                System.out.println(getName() + " is yielding...");
                Thread.yield();
            }
        }
    }
}
public class yielddemo {
    public static void main(String[] args) {
        MyThread1 t1 = new MyThread1();
        MyThread1 t2 = new MyThread1();

        t1.setName("Thread 1");
        t2.setName("Thread 2");

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

```
Output: Thread 2 - 1
Thread 2 - 2
Thread 2 - 3
Thread 1 - 1
Thread 1 - 2
Thread 1 - 3
Thread 2 is yielding...
Thread 1 is yielding...
Thread 2 - 4
Thread 2 - 5
Thread 1 - 4
Thread 1 - 5
```

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

```
class EvenPrinter implements Runnable {
    public void run() {
        for (int i = 2; i <= 10; i += 2) {
            System.out.println("Even: " + i);
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
        }
    }
}
class OddPrinter implements Runnable {
    public void run() {
        for (int i = 1; i <= 9; i += 2) {
            System.out.println("Odd: " + i);
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class evenoddthreads {
    public static void main(String[] args) {
        Thread evenThread = new Thread(new EvenPrinter());
        Thread oddThread = new Thread(new OddPrinter());
        evenThread.start();
        oddThread.start();
    }
}

Output: Even: 2

Odd: 1

Even: 4

Odd: 3

Odd: 5

Even: 6

Even: 8

Odd: 7

Odd: 9

Even: 10
```

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

```
class MyThread5 extends Thread {
    public MyThread5(String name) {
        super(name);
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
```

```
            System.out.println(getName() + " - Count: " + i);
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class threadpriioritydemo {
    public static void main(String[] args) {
        MyThread5 t1 = new MyThread5("Thread 1");
        MyThread5 t2 = new MyThread5("Thread 2");
        MyThread5 t3 = new MyThread5("Thread 3");
        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.NORM_PRIORITY);
        t3.setPriority(Thread.MAX_PRIORITY);
        t1.start();
        t2.start();
        t3.start();
    }
}


Output: Thread 3 - Count: 1

Thread 2 - Count: 1

Thread 1 - Count: 1

Thread 3 - Count: 2

Thread 2 - Count: 2

Thread 1 - Count: 2

Thread 2 - Count: 3

Thread 3 - Count: 3

Thread 1 - Count: 3

Thread 3 - Count: 4

Thread 2 - Count: 4

Thread 1 - Count: 4

Thread 3 - Count: 5

Thread 2 - Count: 5

Thread 1 - Count: 5
```

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

```
class MyThread2 extends Thread {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(getName() + " - " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class threadjoindemo {
    public static void main(String[] args) {
        MyThread2 t = new MyThread2();
        t.setName("Worker Thread");
        t.start();
        try {
            t.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Main thread continues after Worker Thread finishes.");
    }
}

 Output: Worker Thread - 1

Worker Thread - 2

Worker Thread - 3

Main thread continues after Worker Thread finishes.
```

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

```
class StoppableThread extends Thread {
    private volatile boolean running = true;
    public void run() {
        int count = 1;
        while (running) {
            System.out.println("Thread running: " + count++);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Thread stopped.");
    }
    public void stopRunning() {
        running = false;
    }
```

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

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        t.stopRunning();
    }
}
Output: Thread running: 1
Thread running: 2
Thread running: 3
Thread running: 4
Thread stopped.

Output: Thread running: 1

Thread running: 2

Thread running: 3

Thread running: 4

Thread stopped.
```

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

```
class Counter {
    int count = 0;
    public void increment() {
        count++;
    }
}
class MyThread3 extends Thread {
    Counter counter;
    MyThread3(Counter counter) {
        this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}
public class Raceconditiondemo {
    public static void main(String[] args) {
```

```
        Counter counter = new Counter();
        MyThread3 t1 = new MyThread3(counter);
        MyThread3 t2 = new MyThread3(counter);
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final Counter Value: " + counter.count);
    }
}
Output: Final Counter Value: 1759
```

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

```
class Counter1 {
    int count = 0;
    public synchronized void increment() {
        count++;
    }
}
class MyThread4 extends Thread {
    Counter1 counter;
    MyThread4(Counter1 counter) {
        this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}
public class racecon {
    public static void main(String[] args) {
        Counter1 counter = new Counter1();
        MyThread4 t1 = new MyThread4(counter);
        MyThread4 t2 = new MyThread4(counter);
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final Counter Value: " + counter.count);
    }
}
Output: Final Counter Value: 2000
```

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

```
class Counter3 {
    int count = 0;
    public void increment() {
        synchronized (this) {
            count++;
        }
    }
}
class MyThreadt extends Thread {
    Counter3 counter;
    MyThreadt(Counter3 counter) {
        this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}
public class synchronizeblock {
    public static void main(String[] args) {
        Counter3 counter = new Counter3();
        MyThreadt t1 = new MyThreadt(counter);
        MyThreadt t2 = new MyThreadt(counter);
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final Counter Value: " + counter.count);
    }
}
Output: Final Counter Value: 2000
```

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

```
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() + " withdrew " +
amount +
                               " | New Balance: " + balance);
        } else {
            System.out.println(Thread.currentThread().getName() + " tried to
withdraw " + amount +
                               " | Not enough balance!");
        }
    }
}
class DepositThread extends Thread {
    BankAccount account;
    DepositThread(BankAccount account) {
        this.account = account;
    }
    public void run() {
        account.deposit(500);
    }
}
class WithdrawThread extends Thread {
    BankAccount account;
    WithdrawThread(BankAccount account) {
        this.account = account;
    }
    public void run() {
        account.withdraw(700);
    }
}
public class Bankacc {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        Thread t1 = new DepositThread(account);
        Thread t2 = new WithdrawThread(account);
        Thread t3 = new WithdrawThread(account);
        t1.setName("Thread-Deposit");
        t2.setName("Thread-Withdraw1");
        t3.setName("Thread-Withdraw2");
        t1.start();
        t2.start();
        t3.start();
    }
}


Output: Thread-Deposit deposited 500 | New Balance: 1500

Thread-Withdraw2 withdrew 700 | New Balance: 800

Thread-Withdraw1 withdrew 700 | New Balance: 100
```

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

```
class SharedData {
```

```
    private int data;
    private boolean hasData = false;
    public synchronized void produce(int value) {
        while (hasData) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        data = value;
        System.out.println("Produced: " + value);
        hasData = true;
        notify();
    }
    public synchronized void consume() {
        while (!hasData) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Consumed: " + data);
        hasData = false;
        notify();
    }
}
class Producer extends Thread {
    private SharedData shared;

    Producer(SharedData shared) {
        this.shared = shared;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            shared.produce(i);
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        }
    }
}
class Consumer extends Thread {
    private SharedData shared;

    Consumer(SharedData shared) {
        this.shared = shared;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            shared.consume();
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        }
    }
}

public class producerconsumer {
    public static void main(String[] args) {
        SharedData shared = new SharedData();
        Producer producer = new Producer(shared);
```

```
        Consumer consumer = new Consumer(shared);
        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.

```
class SharedPrinter {
    private boolean letterTurn = true;
    public synchronized void printLetter(char letter) {
        while (!letterTurn) {
            try { wait(); } catch (InterruptedException e) {}
        }
        System.out.print(letter + " ");
        letterTurn = false;
        notify();
    }
    public synchronized void printNumber(int number) {
        while (letterTurn) {
            try { wait(); } catch (InterruptedException e) {}
        }
        System.out.print(number + " ");
        letterTurn = true;
        notify();
    }
}
class LetterThread extends Thread {
    private SharedPrinter printer;

    LetterThread(SharedPrinter printer) {
        this.printer = printer;
    }
    public void run() {
```

```
        for (char ch = 'A'; ch <= 'Z'; ch++) {
            printer.printLetter(ch);
        }
    }
}
class NumberThread extends Thread {
    private SharedPrinter printer;

    NumberThread(SharedPrinter printer) {
        this.printer = printer;
    }
    public void run() {
        for (int num = 1; num <= 26; num++) {
            printer.printNumber(num);
        }
    }
}
public class Alternateprint {
    public static void main(String[] args) {
        SharedPrinter printer = new SharedPrinter();
        LetterThread t1 = new LetterThread(printer);
        NumberThread t2 = new NumberThread(printer);
        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().

```
class Message {
    private String content;
    private boolean hasMessage = false;
    public synchronized String readMessage() {
        while (!hasMessage) {
            try { wait(); } catch (InterruptedException e) {}
        }
        hasMessage = false;
        notifyAll();
        return content;
    }
    public synchronized void writeMessage(String message) {
        while (hasMessage) {
            try { wait(); } catch (InterruptedException e) {}
        }
        content = message;
        hasMessage = true;
        notifyAll();
    }
}
```

```
class Writer extends Thread {
    private Message message;
    Writer(Message message) {
        this.message = message;
    }
    public void run() {
        String[] texts = { "Hello", "This is inter-thread", "communication",
"done!", "bye" };
        for (String text : texts) {
            message.writeMessage(text);
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        }
    }
}
class Reader extends Thread {
    private Message message;

    Reader(Message message) {
        this.message = message;
    }
    public void run() {
        String msg;
        do {
            msg = message.readMessage();
            System.out.println("Read: " + msg);
        } while (!msg.equals("bye"));
    }
}
public class waitnotifyalldemo {
    public static void main(String[] args) {
        Message sharedMessage = new Message();
        Writer writer = new Writer(sharedMessage);
        Reader reader1 = new Reader(sharedMessage);
        Reader reader2 = new Reader(sharedMessage);
        writer.start();
        reader1.start();
        reader2.start();
    }
}

Output: Read: Hello

Read: This is inter-thread

Read: communication

Read: done!

Read: bye
```

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

```
import java.time.LocalTime;
class TimePrinter extends Thread {
```

```
    public void run() {
        while (true) {
            System.out.println("Current Time: " + LocalTime.now());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Daemon thread interrupted.");
            }
        }
    }
}
public class Daemonthreaddemo {
    public static void main(String[] args) {
        TimePrinter daemonThread = new TimePrinter();
        daemonThread.setDaemon(true);
        daemonThread.start();
        for (int i = 1; i <= 5; i++) {
            System.out.println("Main thread working: step " + i);
            try {
                Thread.sleep(1500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Main thread finished. Daemon will stop
automatically.");
    }
}
Output: Main thread working: step 1

Current Time: 19:10:13.606929200

Current Time: 19:10:14.608323400

Main thread working: step 2

Current Time: 19:10:15.609746

Main thread working: step 3

Current Time: 19:10:16.610223600

Current Time: 19:10:17.610564400

Main thread working: step 4

Current Time: 19:10:18.611030400

Main thread working: step 5

Current Time: 19:10:19.611697900

Current Time: 19:10:20.613174900

Main thread finished. Daemon will stop automatically.
```

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

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running...");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Thread finished.");
    }
}
public class isalivedemo {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        System.out.println("Before start: isAlive = " + t1.isAlive());
        t1.start();
        System.out.println("After start: isAlive = " + t1.isAlive());
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("After completion: isAlive = " + t1.isAlive());
    }
}
Output: Before start: isAlive = false

After start: isAlive = true

1

2

3

4

5

After completion: isAlive = false
```

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

```
class MyThread extends Thread {
    public MyThread(ThreadGroup group, String name) {
```

```
        super(group, name);
    }

    public void run() {
        System.out.println(getName() + " is running in " +
getThreadGroup().getName());
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(getName() + " finished.");
    }
}
public class threadgrpdemo {
    public static void main(String[] args) {
        ThreadGroup group = new ThreadGroup("MyGroup");
        MyThread t1 = new MyThread(group, "Thread-1");
        MyThread t2 = new MyThread(group, "Thread-2");
        MyThread t3 = new MyThread(group, "Thread-3");
        t1.start();
        t2.start();
        t3.start();
        group.list();
        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("All threads in the group are done.");
    }
}
Output: Thread-1 is running in MyGroup

Thread-3 is running in MyGroup

Thread-2 is running in MyGroup

java.lang.ThreadGroup[name=MyGroup,maxpri=10]

    Thread[#26,Thread-1,5,MyGroup]

    Thread[#27,Thread-2,5,MyGroup]

    Thread[#28,Thread-3,5,MyGroup]

Thread-1 finished.

Thread-3 finished.

Thread-2 finished.

All threads in the group are done
```

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

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
class MultiplyTask implements Callable<Integer> {
    private int a, b;
    public MultiplyTask(int a, int b) {
        this.a = a;
        this.b = b;
    }
    public Integer call() {
        System.out.println("Calculating " + a + " × " + b);
        return a * b;
    }
}
public class callablefuturedemo {
    public static void main(String[] args) {
        MultiplyTask task = new MultiplyTask(5, 6);
        FutureTask<Integer> future = new FutureTask<>(task);
        Thread t = new Thread(future);
        t.start();
        try {
            Integer result = future.get();
            System.out.println("Result: " + result);
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }
}
Output: Calculating 5 × 6

Result: 30
```