Wrapper classes

1. Check if character is a Digit

```
public class DigitCheck {
    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 StringCompareWrapper {
    public static void main(String[] args) {
        String str1 = "100";
        String str2 = "200";

        Integer num1 = Integer.valueOf(str1);
        Integer num2 = Integer.valueOf(str2);

        int comparison = num1.compareTo(num2);

        if (comparison < 0) {
            System.out.println(str1 + " is less than " + str2);
        } else if (comparison > 0) {
            System.out.println(str1 + " is greater than " + str2);
        } else {
            System.out.println(str1 + " is equal to " + str2);
        }
    }
}
```

Output:

```
100 is less than 200
```

3. Convert using valueof method

```
public class WrapperValueOfDemo {
    public static void main(String[] args) {
        Integer intObj = Integer.valueOf("123");
        System.out.println("Integer value: " + intObj);

        Integer intObj2 = Integer.valueOf(456);
        System.out.println("Integer value from int: " + intObj2);

        Double doubleObj = Double.valueOf("3.14159");
        System.out.println("Double value: " + doubleObj);

        Double doubleObj2 = Double.valueOf(2.718);
        System.out.println("Double value from double: " + doubleObj2);

        Boolean boolObj = Boolean.valueOf("true");
        System.out.println("Boolean value: " + boolObj);


        Boolean boolObj2 = Boolean.valueOf(false);
        System.out.println("Boolean value from boolean: " + boolObj2);
    }
}
```

Output:

```
Integer value: 123
Integer value from int: 456
Double value: 3.14159
Double value from double: 2.718
Boolean value: true
Boolean value from boolean: false
```

4. Create Boolean Wrapper usage

```
public class BooleanWrapperDemo {
    public static void main(String[] args) {
        Boolean boolObj1 = Boolean.valueOf(true);
        Boolean boolObj2 = Boolean.valueOf(false);


        Boolean boolObj3 = Boolean.valueOf("true");
        Boolean boolObj4 = Boolean.valueOf("TrUe");   // case-insensitive
        Boolean boolObj5 = Boolean.valueOf("false");
        Boolean boolObj6 = Boolean.valueOf("yes");


        boolean primitiveBool = boolObj1.booleanValue();


        System.out.println("boolObj1 == boolObj3: " + (boolObj1 == boolObj3));  // likely true due to caching
        System.out.println("boolObj1.equals(boolObj3): " + boolObj1.equals(boolObj3));

        System.out.println("boolObj1: " + boolObj1);
        System.out.println("boolObj5: " + boolObj5);
        System.out.println("boolObj6 (from 'yes'): " + boolObj6);

        boolean parsedBool = Boolean.parseBoolean("true");
        System.out.println("Parsed boolean primitive: " + parsedBool);
    }
}
```

Output:

```
boolObj1 == boolObj3: true
boolObj1.equals(boolObj3): true
boolObj1: true
boolObj5: false
boolObj6 (from 'yes'): false
Parsed boolean primitive: true
```

5. Convert null to wrapper classes

```
public class NullToWrapperDemo {
    public static void main(String[] args) {
        Integer intObj = null;
        Boolean boolObj = null;

        System.out.println("intObj: " + intObj);
        System.out.println("boolObj: " + boolObj);

        try {
            int primitiveInt = intObj;
            System.out.println("primitiveInt: " + primitiveInt);
        } catch (NullPointerException e) {
            System.out.println("NullPointerException on unboxing Integer!");
        }
        try {
            boolean primitiveBool = boolObj;
            System.out.println("primitiveBool: " + primitiveBool);
        } catch (NullPointerException e) {
            System.out.println("NullPointerException on unboxing Boolean!");
        }


        if (intObj != null) {
            int primitiveIntSafe = intObj;
            System.out.println("Safe unboxed int: " + primitiveIntSafe);
        } else {
            System.out.println("intObj is null, cannot unbox safely.");
        }
    }
}
```

```
Output:
intObj: null
boolObj: null
NullPointerException on unboxing Integer!
NullPointerException on unboxing Boolean!
intObj is null, cannot unbox safely.
```

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.

```
public class IntPassByValueDemo {


    public static void changeValue(int number) {

        number = 100;

        System.out.println("Inside method, number = " + number);

    }


    public static void main(String[] args) {

        int original = 50;
        System.out.println("Before method call, original = " + original);

        changeValue(original);

        System.out.println("After method call, original = " + original);
    }
}
```

Output:

```
Before method call, original = 50
Inside method, number = 100
After method call, original = 50
```

2. 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
```

3. 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 PrimitivePassDemo {

    public static void modifyPrimitives(int i, double d, boolean b) {
        i = 100;
        d = 99.99;
        b = !b;

        System.out.println("Inside method:");
        System.out.println("i = " + i);
        System.out.println("d = " + d);
        System.out.println("b = " + b);
    }

    public static void main(String[] args) {
        int intVar = 10;
        double doubleVar = 20.5;
        boolean boolVar = true;

        System.out.println("Before method call:");
        System.out.println("intVar = " + intVar);
        System.out.println("doubleVar = " + doubleVar);
        System.out.println("boolVar = " + boolVar);

        modifyPrimitives(intVar, doubleVar, boolVar);

        System.out.println("After method call:");
        System.out.println("intVar = " + intVar);
        System.out.println("doubleVar = " + doubleVar);
        System.out.println("boolVar = " + boolVar);
    }
}
```

Output:

```
Before method call:
intVar = 10
doubleVar = 20.5
boolVar = true
Inside method:
i = 100
d = 99.99
b = false
After method call:
intVar = 10
doubleVar = 20.5
boolVar = 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 modifyLength(Box box, int newLength) {
        box.length = newLength;
        System.out.println("Inside method, length = " + box.length);
    }


    public static void main(String[] args) {
        Box myBox = new Box(10);
        System.out.println("Before method call, length = " + myBox.length);


        modifyLength(myBox, 50);


        System.out.println("After method call, length = " + myBox.length);
    }
}
```

Output:

```
Before method call, length = 10
```

```
Inside method, 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.

```
class Person {
  String name;
  int age;

  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}

public class ObjectPassDemo {

  public static void updatePerson(Person p) {
    p.name = "Ajay";
    p.age = 30;
    System.out.println("Inside method: " + p.name + ", " + p.age);
  }

  public static void main(String[] args) {
    Person person = new Person("vijay", 25);

    System.out.println("Before method call: " + person.name + ", " + person.age);

    updatePerson(person);
```

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

```
Output:
Before method call: vijay, 25
Inside method: Ajay, 30
After method call: Ajay, 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.

```
class Student {
    String name;
    int marks;

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

    void updateMarks(int newMarks) {
        this.marks = newMarks;
    }
}

public class StudentDemo {
    public static void main(String[] args) {
        Student student = new Student("ajay", 75);
        System.out.println("Before update: " + student.name + " has marks " + student.marks);
        student.updateMarks(90);

        System.out.println("After update: " + student.name + " has marks " + student.marks);
    }
}
```

Output:

```
Before update: ajay has marks 75
After update: ajay 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 Box {
    int length;

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

public class CallByValueDemo {

    public static void reassignBox(Box box) {
        box = new Box(999);
        System.out.println("Inside reassignBox: box.length = " + box.length);
    }
    public static void modifyBox(Box box) {
        box.length = 123;
        System.out.println("Inside modifyBox: box.length = " + box.length);
    }

    public static void main(String[] args) {
        Box myBox = new Box(10);

        System.out.println("Before reassignBox: myBox.length = " + myBox.length);
        reassignBox(myBox);
        System.out.println("After reassignBox: myBox.length = " + myBox.length);
        System.out.println();

        System.out.println("Before modifyBox: myBox.length = " + myBox.length);
        modifyBox(myBox);
        System.out.println("After modifyBox: myBox.length = " + myBox.length);
    }
}
```

Output:

```
Before reassignBox: myBox.length = 10
Inside reassignBox: box.length = 999
After reassignBox: myBox.length = 10

Before modifyBox: myBox.length = 10
Inside modifyBox: box.length = 123
After modifyBox: myBox.length = 123
```

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 ReferenceAssignDemo {

    public static void assignNewCar(Car car) {
        car = new Car("Tesla Model S");
        System.out.println("Inside method, car model: " + car.model);
    }

    public static void main(String[] args) {
        Car myCar = new Car("Toyota Corolla");
        System.out.println("Before method call, myCar model: " + myCar.model);

        assignNewCar(myCar);

        System.out.println("After method call, myCar model: " + myCar.model);
    }
}
```

Output:

```
Before method call, myCar model: Toyota Corolla
Inside method, car model: Tesla Model S
```

```
After method call, myCar model: Toyota Corolla
```

---

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

```
public class PrimitiveExample {

    public static void changeValue(int num) {
        num = 100;
        System.out.println("Inside method: num = " + num);
    }

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

        changeValue(number);

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

Output:

```
Before method call: number = 50
Inside method: num = 100
After method call: number = 50
```

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

```
class IntWrapper {
    int value;

    IntWrapper(int value) {
        this.value = value;
    }
}

public class CallByReferenceDemo {

    public static void modifyValue(IntWrapper wrapper) {
        wrapper.value = 999;
        System.out.println("Inside method: wrapper.value = " + wrapper.value);
    }

    public static void main(String[] args) {
        IntWrapper myInt = new IntWrapper(10);
        System.out.println("Before method call: myInt.value = " + myInt.value);

        modifyValue(myInt);

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

Output:

```
Before method call: myInt.value = 10
Inside method: wrapper.value = 999
```

```
            After method call: myInt.value = 999
```

MultiThreading

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

```
class NumberThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        NumberThread t = new NumberThread();
        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("Current thread: " + Thread.currentThread().getName());
    }
}

public class RunnableDemo {
    public static void main(String[] args) {
        MyRunnable task = new MyRunnable();
        Thread thread = new Thread(task);

        thread.start();
    }
}
```

Output:

```
Current thread: Thread-0
```

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

```
class MessagePrinter implements Runnable {
    private String message;

    public 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) {
                System.out.println("Thread interrupted");
            }
        }
    }
}
public class TwoThreadsDemo {
    public static void main(String[] args) {
        Thread thread1 = new Thread(new MessagePrinter("Hello from Thread 1"));
        Thread thread2 = new Thread(new MessagePrinter("Greetings from Thread 2"));

        thread1.start();
        thread2.start();
    }
}
```

Output:

```
Hello from Thread 1- 1
Greetings from Thread 2- 1
Hello from Thread 1- 2
Greetings from Thread 2- 2
...
```

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);
            } catch (InterruptedException e) {
                System.out.println("Thread was interrupted");
            }
        }
    }
}
```

Output:

```
1
2
3
```

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

```
class YieldDemo implements Runnable {
    private String name;

    public YieldDemo(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(name + "- iteration " + i);
            if (i == 3) {
                System.out.println(name + " is yielding...");
                Thread.yield();
            }
        }
    }
}

public class ThreadYieldExample {
    public static void main(String[] args) {
        Thread t1 = new Thread(new YieldDemo("Thread 1"));
        Thread t2 = new Thread(new YieldDemo("Thread 2"));

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

Output:

```
Thread 1- iteration 1
Thread 1- iteration 2
Thread 1- iteration 3
Thread 1 is yielding...
Thread 2- iteration 1
Thread 2- iteration 2
Thread 2- iteration 3
Thread 2 is yielding...
Thread 1- iteration 4
Thread 1- iteration 5
Thread 2- iteration 4
```

Thread 2- iteration 5

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

```
class NumberPrinter {
    private final int max;
    private int number = 1;

    public NumberPrinter(int max) {
        this.max = max;
    }

    public synchronized void printOdd() {
        while (number <= max) {
            if (number % 2 == 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            } else {
                System.out.println("Odd Thread: " + number);
                number++;
                notify();
            }
        }
    }

    public synchronized void printEven() {
        while (number <= max) {
            if (number % 2 != 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            } else {
                System.out.println("Even Thread: " + number);
                number++;
                notify();
            }
        }
    }
}

public class EvenOddThreadDemo {
    public static void main(String[] args) {
        NumberPrinter printer = new NumberPrinter(10);

        Thread oddThread = new Thread(printer::printOdd);
        Thread evenThread = new Thread(printer::printEven);

        oddThread.start();
        evenThread.start();
    }
}
```

Output:

```
Odd Thread: 1
```

```
Even Thread: 2
Odd Thread: 3
Even Thread: 4
Odd Thread: 5
Even Thread: 6
Odd Thread: 7
Even Thread: 8
Odd Thread: 9
Even Thread: 10
```

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

```
class PriorityThread extends Thread {
  public PriorityThread(String name) {
    super(name);
  }
  public void run() {
    System.out.println(getName() + " started with priority " + getPriority());
    for (int i = 1; i <= 3; i++) {
      System.out.println(getName() + " is running iteration " + i);
      try {
        Thread.sleep(300);
      } catch (InterruptedException e) {
        System.out.println(getName() + " interrupted.");
      }
    }
    System.out.println(getName() + " finished.");
  }
}
```

```
public class ThreadPriorityDemo {
    public static void main(String[] args) {
        PriorityThread t1 = new PriorityThread("Thread 1");
        PriorityThread t2 = new PriorityThread("Thread 2");
        PriorityThread t3 = new PriorityThread("Thread 3");
        t1.setPriority(Thread.MIN_PRIORITY);   // 1
        t2.setPriority(Thread.NORM_PRIORITY);  // 5
        t3.setPriority(Thread.MAX_PRIORITY);   // 10

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

Output:

```
Thread 1 started with priority 1
Thread 2 started with priority 5
Thread 3 started with priority 10
Thread 1 is running iteration 1
Thread 3 is running iteration 1
Thread 2 is running iteration 1
Thread 1 is running iteration 2
Thread 3 is running iteration 2
Thread 2 is running iteration 2
Thread 3 is running iteration 3
Thread 1 is running iteration 3
Thread 2 is running iteration 3
Thread 3 finished.
Thread 1 finished.

Thread 2 finished.
```

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

```
class WorkerThread extends Thread {
    public void run() {
        System.out.println("Worker thread started.");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            System.out.println("Worker thread interrupted.");
        }
        System.out.println("Worker thread finished.");
    }
}

public class ThreadJoinDemo {
    public static void main(String[] args) {
        WorkerThread worker = new WorkerThread();
        worker.start();

        System.out.println("Main thread waiting for worker to finish...");

        try {
            worker.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted while waiting.");
        }
        System.out.println("Worker has finished. Main thread resumes.");
    }
}
```

Output:

```
Worker thread started.
Main thread waiting for worker to finish...
Worker thread finished.
Worker has finished. Main thread resumes.
```

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

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

    public void stopRunning() {
        running = false;
    }
    public void run() {
        int count = 1;
        while (running) {
            System.out.println("Thread running: count = " + count++);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted.");
                running = false;
            }
        }
        System.out.println("Thread stopped.");
    }
}

public class ThreadStopDemo {
    public static void main(String[] args) throws InterruptedException {
        StoppableThread t = new StoppableThread();
        t.start();

        Thread.sleep(3000);

        System.out.println("Requesting thread to stop...");
        t.stopRunning();

        t.join();
        System.out.println("Main thread exiting.");
    }
}
```

Output:

```
Thread running: count = 1
Thread running: count = 2
Thread running: count = 3
Thread running: count = 4
Thread running: count = 5
Requesting thread to stop...
Thread stopped.
Main thread exiting.
```

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

```
class Counter {
    public int count = 0;

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

class CounterThread extends Thread {
    private Counter counter;

    public CounterThread(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) throws InterruptedException {
        Counter counter = new Counter();

        Thread t1 = new CounterThread(counter);
        Thread t2 = new CounterThread(counter);
        t1.start();
        t2.start();

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

        System.out.println("Final count (expected 2000): " + counter.count);
    }
}
```

Output:

```
Final count (expected 2000): 1785
```

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

```
class Counter {
    private int count = 0;
    private final Object lock = new Object();  // Lock object for synchronization

    public void increment() {
        synchronized (lock) {
            count++;  // Only one thread can execute this block at a time
        }
    }

    public int getCount() {
        synchronized (lock) {
            return count;
        }
    }
}

class CounterThread extends Thread {
    private Counter counter;

    public CounterThread(Counter counter) {
        this.counter = counter;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}

public class SynchronizedBlockDemo {
    public static void main(String[] args) throws InterruptedException {
        Counter counter = new Counter();

        Thread t1 = new CounterThread(counter);
        Thread t2 = new CounterThread(counter);

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

        t1.join();
        t2.join();
        System.out.println("Final count (expected 2000): " + counter.getCount());
    }
}
```

Output:

```
Final count (expected 2000): 2000
```

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

```
class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
    public synchronized void deposit(double amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited " + amount + ", 
balance: " + balance);
        notifyAll();
    }

    public synchronized void withdraw(double amount) {
        while (balance < amount) {
            try {
                System.out.println(Thread.currentThread().getName() + " waiting to withdraw " + 
amount + ", balance: " + balance);
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println(Thread.currentThread().getName() + " interrupted.");
                return;
            }
        }
        balance-= amount;
        System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ", 
balance: " + balance);
    }

    public synchronized double getBalance() {
        return balance;
    }
}

class DepositThread extends Thread {
    private BankAccount account;
    private double amount;

    public DepositThread(BankAccount account, double amount, String name) {
        super(name);
        this.account = account;
        this.amount = amount;
    }
    public void run() {
        account.deposit(amount);
    }
}

class WithdrawThread extends Thread {
    private BankAccount account;
    private double amount;
    public WithdrawThread(BankAccount account, double amount, String name) {
        super(name);
        this.account = account;
        this.amount = amount;
    }
    public void run() {
        account.withdraw(amount);
    }
}

public class BankAccountDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);

        Thread t1 = new WithdrawThread(account, 1500, "Withdraw-1");
        Thread t2 = new DepositThread(account, 700, "Deposit-1");
        Thread t3 = new WithdrawThread(account, 300, "Withdraw-2");

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

        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        System.out.println("Final balance: " + account.getBalance());
    }
}
```

Output:

```
Withdraw-1 waiting to withdraw 1500.0, balance: 1000.0

Deposit-1 deposited 700.0, balance: 1700.0

Withdraw-1 withdrew 1500.0, balance: 200.0

Withdraw-2 withdrew 300.0, balance:-100.0

Final balance:-100.0
```

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

```
class Drop {
    private final Queue<Integer> buffer = new LinkedList<>();
    private final int capacity;

    public Drop(int capacity) {
        this.capacity = capacity;
    }
    public synchronized void put(int value) throws InterruptedException {
        while (buffer.size() == capacity) {
            wait();
        }
        buffer.offer(value);
        System.out.println("Produced: " + value);
        notify();  // notify consumer waiting
    }
    public synchronized int take() throws InterruptedException {
        while (buffer.isEmpty()) {
            wait();  // wait if buffer is empty
        }
        int value = buffer.poll();
        System.out.println("Consumed: " + value);
        notify();
        return value;
    }
}

class Producer implements Runnable {
    private final Drop drop;

    public Producer(Drop drop) {
        this.drop = drop;
    }
    public void run() {
        for (int i = 1; i <= 10; i++) {
            try {
                drop.put(i);
                Thread.sleep(500);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
```

```
}

class Consumer implements Runnable {
    private final Drop drop;

    public Consumer(Drop drop) {
        this.drop = drop;
    }
    public void run() {
        for (int i = 1; i <= 10; i++) {
            try {
                drop.take();
                Thread.sleep(800);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

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

        Thread producer = new Thread(new Producer(drop));
        Thread consumer = new Thread(new Consumer(drop));

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

Output:

```
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Consumed: 2
Produced: 4
Consumed: 3
Produced: 5
Produced: 6
Consumed: 4
Produced: 7
Consumed: 5
...
```

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

```
class Alternator {
    private boolean letterTurn = true;

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

class LetterThread extends Thread {
    private final Alternator alternator;

    public LetterThread(Alternator alternator) {
        this.alternator = alternator;
    }
    public void run() {
        try {
            for (char c = 'A'; c <= 'Z'; c++) {
                alternator.printLetter(c);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

class NumberThread extends Thread {
    private final Alternator alternator;


    public NumberThread(Alternator alternator) {
        this.alternator = alternator;
    }
    public void run() {
        try {
            for (int i = 1; i <= 26; i++) {
                alternator.printNumber(i);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}


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


        Thread letterThread = new LetterThread(alternator);
        Thread numberThread = new NumberThread(alternator);


        letterThread.start();
        numberThread.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
```

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

```
class MessageQueue {
    private String message;
    private boolean empty = true;

    public synchronized void put(String msg) throws InterruptedException {
        while (!empty) {
            wait();
        }
        message = msg;
        empty = false;
        System.out.println("Produced: " + msg);
        notifyAll();
    }


    public synchronized String take() throws InterruptedException {
        while (empty) {
            wait();
        }
        String msg = message;
        empty = true;
        System.out.println(Thread.currentThread().getName() + " consumed: " + msg);
        notifyAll();  // notify producer and other consumers
        return msg;
    }
}
```

```
class Producer implements Runnable {
    private final MessageQueue queue;

    public Producer(MessageQueue queue) {
        this.queue = queue;
    }
    public void run() {
        String[] messages = { "Hello", "World", "Java", "Threads", "Done" };
        try {
            for (String msg : messages) {
                queue.put(msg);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

class Consumer implements Runnable {
    private final MessageQueue queue;

    public Consumer(MessageQueue queue) {
        this.queue = queue;
    }
    public void run() {
        try {
            while (true) {
                String msg = queue.take();
```

```java
                if ("Done".equals(msg)) {
                    break;
                }
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        System.out.println(Thread.currentThread().getName() + " exiting.");
    }
}

public class WaitNotifyAllDemo {
    public static void main(String[] args) {
        MessageQueue queue = new MessageQueue();

        Thread producer = new Thread(new Producer(queue), "Producer");
        Thread consumer1 = new Thread(new Consumer(queue), "Consumer-1");
        Thread consumer2 = new Thread(new Consumer(queue), "Consumer-2");

        producer.start();
        consumer1.start();
        consumer2.start();
    }
}
```

Output:

```
Produced: Hello
Consumer-1 consumed: Hello
Consumer-2 consumed: Hello
```

```
Produced: World
Consumer-1 consumed: World
Consumer-2 consumed: World
...
Produced: Done
Consumer-1 consumed: Done
Consumer-1 exiting.
Consumer-2 consumed: Done
Consumer-2 exiting.
```

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

```java
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

class TimePrinter extends Thread {
    public void run() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
        try {
            while (true) {
                System.out.println("Current time: " + LocalTime.now().format(formatter));
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Daemon thread interrupted");
        }
    }
}
```

```
public class DaemonThreadDemo {
    public static void main(String[] args) throws InterruptedException {
        TimePrinter daemonThread = new TimePrinter();
        daemonThread.setDaemon(true);
        daemonThread.start();

        System.out.println("Main thread running for 5 seconds...");
        Thread.sleep(5000);

        System.out.println("Main thread finished. Daemon thread will stop automatically.");
    }
}
```

Output:

```
Main thread running for 5 seconds...
Current time: 12:30:15
Current time: 12:30:16
Current time: 12:30:17
Current time: 12:30:18
Current time: 12:30:19
Main thread finished. Daemon thread will stop automatically.
```

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

```
class SimpleThread extends Thread {
    public void run() {
        System.out.println(getName() + " started.");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            System.out.println(getName() + " interrupted.");
        }
        System.out.println(getName() + " finished.");
    }
}

public class ThreadIsAliveDemo {
    public static void main(String[] args) throws InterruptedException {
        SimpleThread t = new SimpleThread();
        System.out.println("Before start: isAlive() = " + t.isAlive());

        t.start();

        System.out.println("Just after start: isAlive() = " + t.isAlive());

        Thread.sleep(1000);
        System.out.println("After 1 second: isAlive() = " + t.isAlive());

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

Output:

```
Before start: isAlive() = false
SimpleThread-0 started.
Just after start: isAlive() = true
After 1 second: isAlive() = true
SimpleThread-0 finished.
After join: isAlive() = false
```

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

```
class Task implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName() + " started in group: "
                        + Thread.currentThread().getThreadGroup().getName());
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            System.out.println(Thread.currentThread().getName() + " interrupted.");
        }
        System.out.println(Thread.currentThread().getName() + " finished.");
    }
}

public class ThreadGroupDemo {
    public static void main(String[] args) throws InterruptedException {
        ThreadGroup group = new ThreadGroup("MyThreadGroup");
        Thread t1 = new Thread(group, new Task(), "Thread-1");
        Thread t2 = new Thread(group, new Task(), "Thread-2");
        Thread t3 = new Thread(group, new Task(), "Thread-3");
        t1.start();
        t2.start();
        t3.start();
        System.out.println("Active threads in group: " + group.activeCount());
        Thread[] threads = new Thread[group.activeCount()];
        group.enumerate(threads);
        System.out.println("Threads in group:");
        for (Thread t : threads) {
            System.out.println("- " + t.getName());
        }
        Thread.sleep(1000);
        System.out.println("Interrupting all threads in group...");
        group.interrupt();
        t1.join();
        t2.join();
        t3.join();

        System.out.println("All threads in the group finished.");
    }
}
```

Output:

```
Thread-1 started in group: MyThreadGroup
Thread-2 started in group: MyThreadGroup
Thread-3 started in group: MyThreadGroup
Active threads in group: 3
Threads in group:
- Thread-1
- Thread-2
- Thread-3
Interrupting all threads in group...
Thread-1 interrupted.
Thread-2 interrupted.
Thread-3 interrupted.
Thread-1 finished.
Thread-2 finished.
Thread-3 finished.
All threads in the group finished.
```

19 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.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;


class MultiplicationTask implements Callable<Integer> {

    private final int a;

    private final int b;


    public MultiplicationTask(int a, int b) {

        this.a = a;

        this.b = b;

    }

    public Integer call() {

        int result = a * b;

        System.out.println("Calculating: " + a + " * " + b);

        return result;

    }

}


public class CallableFutureDemo {

    public static void main(String[] args) {


        ExecutorService executor = Executors.newSingleThreadExecutor();

        MultiplicationTask task = new MultiplicationTask(6, 7);
        Future<Integer> futureResult = executor.submit(task);

        try {
            int result = futureResult.get();
            System.out.println("Result: " + result);
        } catch (InterruptedException | ExecutionException e) {
            System.out.println("Exception occurred: " + e.getMessage());
        } finally {
            executor.shutdown();
        }
    }
}
```

Output:

```
Calculating: 6 * 7
Result: 42
```