Wrapper Classes

1. Check if character is a Digit

```
class WrapperDemo1 {
   public static void main(String[] args) {
      char ch = '5';
      boolean result = Character.isDigit(ch);
      System.out.println(ch + " is digit? " + result);
   }
}
```

Output:

5 is digit? true

2. Compare two Strings

```
class WrapperDemo2 {
   public static void main(String[] args) {
      String s1 = "hello";
      String s2 = "hello";
      System.out.println("Using equals(): " + s1.equals(s2));
      System.out.println("Using == : " + (s1 == s2));
   }
}
```

Output:

Using equals(): true

Using == : true

3. Convert using valueOf method

```
class WrapperDemo3 {
   public static void main(String[] args) {
      int num = 123;
      String str = String.valueOf(num);
      System.out.println("String from int: " + str);
   }
}
```

Output:

String from int: 123

4. Create Boolean Wrapper usage

```
class WrapperDemo4 {
   public static void main(String[] args) {
      Boolean b1 = Boolean.valueOf(true);
      Boolean b2 = Boolean.TRUE;
      System.out.println("Boolean b1: " + b1);
      System.out.println("Boolean b2: " + b2);
   }
}
```

Output:

Boolean b1: true

Boolean b2: true

5. Convert null to wrapper classes

```
class WrapperDemo5 {
   public static void main(String[] args) {
      Integer num = null;
      try {
         int n = num; // causes NullPointerException
         System.out.println(n);
      } catch (NullPointerException e) {
         System.out.println("NullPointerException caught");
      }
   }
}
```

Output:

NullPointerException caught

Pass By Value and Pass By Reference

1. Method accepts int and tries to change value

```
class PassByValue1 {
```

```
    void change(int x) {
        x = 100;
    }
    public static void main(String[] args) {
        PassByValue1 obj = new PassByValue1();
        int a = 50;
        System.out.println("Before change: " + a);
        obj.change(a);
        System.out.println("After change: " + a);
    }
}
```

Output:

```
Before change: 50
After change: 50
```

2. Method swaps two integers (no effect outside method)

```
class PassByValue2 {
    void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
    }
    public static void main(String[] args) {
        PassByValue2 obj = new PassByValue2();
        int x = 10, y = 20;
        System.out.println("Before swap: x=" + x + ", y=" + y);
        obj.swap(x, y);
        System.out.println("After swap: x=" + x + ", y=" + y);
    }
}
```

Output:

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

After swap: x=10, y=20

3. Pass primitives and change inside method

```
class PassByValue3 {
   void change(int a, double b) {
      a = 100;
      b = 99.99;
   }
   public static void main(String[] args) {
      PassByValue3 obj = new PassByValue3();
      int x = 10;
      double y = 20.5;
      System.out.println("Before change: x=" + x + ", y=" + y);
      obj.change(x, y);
      System.out.println("After change: x=" + x + ", y=" + y);
   }
}
```

Output:

Before change: x=10, y=20.5

After change: x=10, y=20.5

Call By Reference (Using Objects)

4. Box class length modified by method

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

class PassByRef1 {
   void modifyLength(Box b) {
      b.length = 100;
```

```
    }
    public static void main(String[] args) {
        Box box = new Box(50);
        System.out.println("Before modify: " + box.length);
        new PassByRef1().modifyLength(box);
        System.out.println("After modify: " + box.length);
    }
}
```

Output:

Before modify: 50

After modify: 100

5. Modify object internal fields

```
class User {
    String username;
    String password;
}


class PassByRef2 {
    void update(User u) {
        u.username = "admin";
        u.password = "pass123";
    }
    public static void main(String[] args) {
        User user = new User();
        user.username = "guest";
        user.password = "guest123";
        System.out.println("Before update: " + user.username + ", " + user.password);
        new PassByRef2().update(user);
        System.out.println("After update: " + user.username + ", " + user.password);
    }
}
```

Output:

Before update: guest, guest123

After update: admin, pass123

6. Student marks update

```
class Student {
   String name;
   int marks;
   Student(String n, int m) {
      name = n;
      marks = m;
   }
}

class PassByRef3 {
   void updateMarks(Student s, int newMarks) {
      s.marks = newMarks;
   }
   public static void main(String[] args) {
      Student stu = new Student("Monalisa", 70);
      System.out.println("Before: " + stu.name + " marks=" + stu.marks);
      new PassByRef3().updateMarks(stu, 90);
      System.out.println("After: " + stu.name + " marks=" + stu.marks);
   }
}
```

Output:

Before: Monalisa marks=70

After: Monalisa marks=90

7. Java is strictly call by value

```
class PassByRef4 {
   void change(User u) {
      u = new User();
```

```
        u.username = "newUser";
    }
    public static void main(String[] args) {
        User user = new User();
        user.username = "oldUser";
        System.out.println("Before: " + user.username);
        new PassByRef4().change(user);
        System.out.println("After: " + user.username);
    }
}
```

Output:

Before: oldUser

After: oldUser

8. Assign new object to reference passed

```
class PassByRef5 {
    void assignNew(User u) {
        u = new User();
        u.username = "assignedUser";
    }
    public static void main(String[] args) {
        User user = new User();
        user.username = "originalUser";
        System.out.println("Before: " + user.username);
        new PassByRef5().assignNew(user);
        System.out.println("After: " + user.username);
    }
}
```

Output:

Before: originalUser

After: originalUser

9. Difference between passing primitive and non-primitive types

```
class PrimitiveNonPrimitive {
    void modify(int a, User u) {
        a = 99;
        u.username = "modified";
    }
    public static void main(String[] args) {
        int x = 10;
        User user = new User();
        user.username = "original";
        System.out.println("Before: x=" + x + ", user=" + user.username);
        new PrimitiveNonPrimitive().modify(x, user);
        System.out.println("After: x=" + x + ", user=" + user.username);
    }
}
```

Output:

Before: x=10, user=original

After: x=10, user=modified

10. Simulate call by reference using array

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

Output:

Before: 10

After: 999

MultiThreading

1. Thread by extending Thread class print 1 to 5

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

Output:

1

2

3

4

5

2. Thread by implementing Runnable prints thread name

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

Output:

Thread: Thread-0

3. Two threads printing different messages 5 times

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

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

Output

```
Hello from Thread 1
Hello from Thread 2
Hello from Thread 1
Hello from Thread 2
...
```

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

```
class SleepDemo extends Thread {
   public void run() {
      for(int i=1; i<=3; i++) {
         System.out.println(i);
         try {
            Thread.sleep(1000);
```

```
            } catch(Exception e) {}
        }
    }
    public static void main(String[] args) {
        new SleepDemo().start();
    }
}
```

Output

```
1
2
3
```

5. Thread.yield() example

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

Output

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

...

6. Two threads print even and odd numbers

```
class EvenThread extends Thread {
    public void run() {
        for(int i=2; i<=10; i+=2) System.out.println("Even: " + i);
    }
}
class OddThread extends Thread {
    public void run() {
        for(int i=1; i<=9; i+=2) System.out.println("Odd: " + i);
    }
}
class EvenOddDemo {
    public static void main(String[] args) {
        new EvenThread().start();
        new OddThread().start();
    }
}
```

Output

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

...

7. Three threads with different priorities

```
class PriorityThread extends Thread {
    PriorityThread(String name) { super(name); }
    public void run() {
        for(int i=1; i<=3; i++)
            System.out.println(getName() + " - " + i);
    }
```

```
}
class PriorityDemo {
   public static void main(String[] args) {
      PriorityThread t1 = new PriorityThread("Low");
      PriorityThread t2 = new PriorityThread("Normal");
      PriorityThread t3 = new PriorityThread("High");

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

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

Output

```
High - 1
Normal - 1
Low - 1
...
```

8. Thread.join() demo

```
class JoinDemo extends Thread {
   public void run() {
      for(int i=1; i<=3; i++) System.out.println(getName() + " - " + i);
   }
}
class JoinExample {
   public static void main(String[] args) throws Exception {
      JoinDemo t1 = new JoinDemo();
      t1.setName("Thread-1");
```

```
        t1.start();
        t1.join();
        System.out.println("Main thread resumes after join");
    }
}
```

Output:

```
Thread-1 - 1
Thread-1 - 2
Thread-1 - 3
Main thread resumes after join
```

9. Stop thread using boolean flag

```
class StopThread extends Thread {
    volatile boolean running = true;
    public void run() {
        while(running) {
            System.out.println("Thread running");
            try { Thread.sleep(500); } catch(Exception e) {}
        }
        System.out.println("Thread stopped");
    }
    public void stopRunning() {
        running = false;
    }
    public static void main(String[] args) throws Exception {
        StopThread t = new StopThread();
        t.start();
        Thread.sleep(1500);
        t.stopRunning();
    }
}
```

Output:

```
Thread running
Thread running
Thread running
Thread stopped
```

10. Multiple threads access shared counter without synchronization

```
class Counter {
   int count = 0;
   void increment() {
      count++;
   }
}

class RaceConditionDemo implements Runnable {
   Counter counter;
   RaceConditionDemo(Counter c) { counter = c; }
   public void run() {
      for(int i=0; i<1000; i++) {
         counter.increment();
      }
   }
   public static void main(String[] args) throws Exception {
      Counter c = new Counter();
      Thread t1 = new Thread(new RaceConditionDemo(c));
      Thread t2 = new Thread(new RaceConditionDemo(c));
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Count: " + c.count);
   }
}
```

Output

Count: 1765

11. Solve above with synchronized

```
class CounterSync {
   int count = 0;
   synchronized void increment() {
      count++;
   }
}

class RaceConditionFix implements Runnable {
   CounterSync counter;
   RaceConditionFix(CounterSync c) { counter = c; }
   public void run() {
      for(int i=0; i<1000; i++) {
         counter.increment();
      }
   }
   public static void main(String[] args) throws Exception {
      CounterSync c = new CounterSync();
      Thread t1 = new Thread(new RaceConditionFix(c));
      Thread t2 = new Thread(new RaceConditionFix(c));
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Count: " + c.count);
   }
}
```

Output:

Count: 2000

12. Synchronized block mutual exclusion

```
class SyncBlock {
   int count = 0;
   void increment() {
      synchronized(this) {
         count++;
      }
   }
}

class SyncBlockDemo implements Runnable {
   SyncBlock sb;
   SyncBlockDemo(SyncBlock sb) { this.sb = sb; }
   public void run() {
      for(int i=0; i<1000; i++) {
         sb.increment();
      }
   }
   public static void main(String[] args) throws Exception {
      SyncBlock sb = new SyncBlock();
      Thread t1 = new Thread(new SyncBlockDemo(sb));
      Thread t2 = new Thread(new SyncBlockDemo(sb));
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Count: " + sb.count);
   }
}
```

Output

```
Count: 2000
```

13. BankAccount with synchronization

```
class BankAccount {
   int balance = 1000;
   synchronized void deposit(int amount) {
      balance += amount;
      System.out.println("Deposited " + amount + ", balance: " + balance);
   }
   synchronized void withdraw(int amount) {
      if(balance >= amount) {
         balance -= amount;
         System.out.println("Withdrew " + amount + ", balance: " + balance);
      } else {
         System.out.println("Insufficient balance");
      }
   }
}

class BankDemo implements Runnable {
   BankAccount acc;
   boolean deposit;
   int amount;
   BankDemo(BankAccount acc, boolean dep, int amt) {
      this.acc = acc;
      deposit = dep;
      amount = amt;
   }
   public void run() {
      if(deposit)
         acc.deposit(amount);
      else
         acc.withdraw(amount);
```

```
    }
    public static void main(String[] args) throws Exception {
        BankAccount acc = new BankAccount();
        Thread t1 = new Thread(new BankDemo(acc, true, 500));
        Thread t2 = new Thread(new BankDemo(acc, false, 800));
        t1.start();
        t2.start();
        t1.join();
        t2.join();
    }
}
```

Output:

```
Deposited 500, balance: 1500
Withdrew 800, balance: 700
```

14. Producer-Consumer problem using wait and notify

```
class Q {
    int n;
    boolean valueSet = false;

    synchronized void put(int n) {
        while(valueSet) {
            try { wait(); } catch(Exception e) {}
        }
        this.n = n;
        System.out.println("Put: " + n);
        valueSet = true;
        notify();
    }

    synchronized int get() {
        while(!valueSet) {
```

```
            try { wait(); } catch(Exception e) {}
        }
        System.out.println("Got: " + n);
        valueSet = false;
        notify();
        return n;
    }
}

class Producer implements Runnable {
    Q q;
    Producer(Q q) { this.q = q; }
    public void run() {
        int i=0;
        while(i<5) q.put(i++);
    }
}

class Consumer implements Runnable {
    Q q;
    Consumer(Q q) { this.q = q; }
    public void run() {
        int val = 0;
        while(val<5) val = q.get();
    }
}

class PCDemo {
    public static void main(String[] args) {
        Q q = new Q();
        new Thread(new Producer(q)).start();
```

```
        new Thread(new Consumer(q)).start();
    }
}
```

Output

```
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
```

15. Thread printing A-Z and 1-26 alternately

```
class PrintAlphaNum {
    boolean letterTurn = true;
    synchronized void printLetter(char c) throws InterruptedException {
        while(!letterTurn) wait();
        System.out.print(c);
        letterTurn = false;
        notify();
    }
    synchronized void printNumber(int n) throws InterruptedException {
        while(letterTurn) wait();
        System.out.print(n);
        letterTurn = true;
        notify();
    }
}
```

```
class LetterThread extends Thread {
   PrintAlphaNum pan;
   LetterThread(PrintAlphaNum pan) { this.pan = pan; }
   public void run() {
      try {
         for(char c='A'; c<='Z'; c++) {
            pan.printLetter(c);
         }
      } catch(InterruptedException e) {}
   }
}

class NumberThread extends Thread {
   PrintAlphaNum pan;
   NumberThread(PrintAlphaNum pan) { this.pan = pan; }
   public void run() {
      try {
         for(int i=1; i<=26; i++) {
            pan.printNumber(i);
         }
      } catch(InterruptedException e) {}
   }
}

class AlphaNumDemo {
   public static void main(String[] args) {
      PrintAlphaNum pan = new PrintAlphaNum();
      new LetterThread(pan).start();
      new NumberThread(pan).start();
   }
}
```

Output:

A1B2C3...Z26

16. Inter-thread communication using wait() and notifyAll()

```
class Message {
   String msg;
   boolean empty = true;

   synchronized void put(String m) throws InterruptedException {
      while(!empty) wait();
      msg = m;
      empty = false;
      notifyAll();
   }

   synchronized String get() throws InterruptedException {
      while(empty) wait();
      empty = true;
      notifyAll();
      return msg;
   }
}

class ProducerComm implements Runnable {
   Message msg;
   ProducerComm(Message m) { msg = m; }
   public void run() {
      String[] msgs = {"Hello", "World", "Java", "Threads", "Bye"};
      try {
         for(String m : msgs) {
            msg.put(m);
            Thread.sleep(500);
```

```
            }
        } catch(Exception e) {}
    }
}

class ConsumerComm implements Runnable {
    Message msg;
    ConsumerComm(Message m) { msg = m; }
    public void run() {
        try {
            for(int i=0; i<5; i++) {
                System.out.println("Received: " + msg.get());
            }
        } catch(Exception e) {}
    }
}

class CommDemo {
    public static void main(String[] args) {
        Message msg = new Message();
        new Thread(new ProducerComm(msg)).start();
        new Thread(new ConsumerComm(msg)).start();
    }
}
```

Output:

```
Received: Hello
Received: World
Received: Java
Received: Threads
Received: Bye
```

17. Daemon thread printing time every second

```
class DaemonThread extends Thread {
   public void run() {
      while(true) {
         System.out.println("Time: " + System.currentTimeMillis());
         try {
            Thread.sleep(1000);
         } catch(Exception e) {}
      }
   }
   public static void main(String[] args) throws Exception {
      DaemonThread t = new DaemonThread();
      t.setDaemon(true);
      t.start();
      Thread.sleep(4000);
      System.out.println("Main thread finished");
   }
}
```

Output

```
Time: 1691618829282
Time: 1691618821029
Time: 1691618822030
Main thread finished
```

18. Thread.isAlive() demo

```
class IsAliveDemo extends Thread {
   public void run() {
      try {
         Thread.sleep(1000);
      } catch(Exception e) {}
   }
   public static void main(String[] args) throws Exception {
      IsAliveDemo t = new IsAliveDemo();
```

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

Output:

Before start: false

After start: true

After join: false

19. Thread group creation and management

```
class ThreadGroupDemo {
    public static void main(String[] args) throws Exception {
        ThreadGroup group = new ThreadGroup("MyGroup");

        Thread t1 = new Thread(group, () -> {
            System.out.println("Thread 1 running in " +
Thread.currentThread().getThreadGroup().getName());
        });
        Thread t2 = new Thread(group, () -> {
            System.out.println("Thread 2 running in " +
Thread.currentThread().getThreadGroup().getName());
        });

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

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

        System.out.println("Active threads in group: " + group.activeCount());
```

```
        group.list();
    }
}
```

Output:

```
Thread 1 running in MyGroup
Thread 2 running in MyGroup
Active threads in group: 0
java.lang.ThreadGroup[name=MyGroup,maxpri=10]
```

20. Thread with Callable and Future

```
import java.util.concurrent.*;
class CallableDemo implements Callable<Integer> {
    public Integer call() {
        int result = 1;
        for(int i=1; i<=5; i++) result *= i;
        return result;
    }
}

class CallableExample {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newSingleThreadExecutor();
        Future<Integer> future = executor.submit(new CallableDemo());
        System.out.println("Factorial: " + future.get());
        executor.shutdown();
    }
}
```

Output:

```
Factorial: 120
```