Wrapper classes

1. Check if character is a Digit

ANS:

```
package Day_7;

public class CheckDigit {
	public static void main(String[] args) {
    char ch = '5';

    if (Character.isDigit(ch)) {
      System.out.println(ch + " is a digit");
    } else {
      System.out.println(ch + " is not a digit");
    }
  }
}
```

Output:

```
5 is a digit
```

2. Compare two Strings

ANS:

```
package Day_7;

public class CompareStrings {
  public static void main(String[] args) {
    String str1 = "Java";
    String str2 = "java";

    System.out.println("Using equals(): " + str1.equals(str2));
    System.out.println("Using equalsIgnoreCase(): " + str1.equalsIgnoreCase(str2));
  }
}
```

Output:

```
Using equals(): false
Using equalsIgnoreCase(): true
```

3. Convert using valueof method

ANS:

```
package Day_7;

public class ValueOfExample {
  public static void main(String[] args) {
    int num = 100;
    String str = String.valueOf(num);

    System.out.println("Integer to String: " + str);

    double d = 55.5;
```

```
        String str2 = String.valueOf(d);
        System.out.println("Double to String: " + str2);
    }
}
```

Output:

```
Integer to String: 100
Double to String: 55.5
```

4. Create Boolean Wrapper usage

ANS:

```
package Day_7;

public class BooleanWrapper {
    public static void main(String[] args) {
        Boolean b1 = Boolean.valueOf(true);
        Boolean b2 = Boolean.valueOf("false");

        System.out.println("b1: " + b1);
        System.out.println("b2: " + b2);
    }
}
```

Output:

```
b1: true
b2: false
```

5. Convert null to wrapper classes

ANS:

```
package Day_7;

public class NullToWrapper {
    public static void main(String[] args) {
        String str = null;

        try {
            Integer num = Integer.valueOf(str); // This will throw exception
            System.out.println(num);
        } catch (NumberFormatException e) {
            System.out.println("Cannot convert null to Integer");
        }
    }
}
```

Output:

```
Cannot convert null to Integer
```

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.

ANS:

```
package Day_7;

public class PassByValue {
   public static void changeValue(int x) {
      x = 20;
   }

   public static void main(String[] args) {
      int num = 10;
      System.out.println("Before: " + num);

      changeValue(num);

      System.out.println("After: " + num);
   }
}
```

Output:

```
Before: 10
After: 10
```

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

ANS:

```
package Day_7;

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

   public static void main(String[] args) {
      int x = 5, y = 10;
      System.out.println("Before: x=" + x + ", y=" + y);

      swap(x, y);

      System.out.println("After: x=" + x + ", y=" + y);
}
}
```

Output:

```
Before: x=5, y=10
Inside method: a=10, b=5
After: x=5, y=10
```

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

ANS:

```
package Day_7;

public class PrimitiveExample {
   public static void change(int value) {
      value = value + 5;
   }

   public static void main(String[] args) {
      int num = 50;
      change(num);
      System.out.println("After method call: " + num);
   }
}
```

Output:

```
After method call: 50
```

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.

ANS:

```
package Day_7;

class Box {
   int length;
}

public class ModifyBox {
   public static void changeLength(Box b) {
      b.length = 20;
   }

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

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

Output:

```
Before: 10
After: 20
```

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.

ANS:

```
package Day_7;

class Person {
   String name;
}

public class ModifyPerson {
   public static void changeName(Person p) {
      p.name = "John";
   }

   public static void main(String[] args) {
      Person per = new Person();
      per.name = "Alice";

      System.out.println("Before: " + per.name);
      changeName(per);
      System.out.println("After: " + per.name);
   }
}
```

Output:

```
Before: Alice
After: John
```

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.

ANS:

```
package Day_7;

class Student {
   String name;
   int marks;
}

public class UpdateMarks {
   public static void update(Student s) {
      s.marks += 10;
   }

   public static void main(String[] args) {
      Student st = new Student();
      st.name = "Raj";
      st.marks = 80;

      System.out.println("Before: " + st.marks);
      update(st);
      System.out.println("After: " + st.marks);
```

```
    }
}
```

Output:

```
Before: 80
After: 90
```

---

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

ANS:

```
package Day_7;

class MyClass {
    int data;
}

public class CallByValueObject {
    public static void changeRef(MyClass obj) {
        obj = new MyClass(); // new object
        obj.data = 50;
    }

    public static void main(String[] args) {
        MyClass mc = new MyClass();
        mc.data = 10;

        changeRef(mc);
        System.out.println("After: " + mc.data);
    }
}
```

Output:

```
After: 10
```

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.

ANS:

```
package Day_7;

class Car {
    String model;
}

public class NewObjectInside {
    public static void change(Car c) {
        c = new Car();
        c.model = "BMW";
    }
```

```
   public static void main(String[] args) {
      Car myCar = new Car();
      myCar.model = "Toyota";

      change(myCar);
      System.out.println("After: " + myCar.model);
   }
}
```

Output:

```
After: Toyota
```

---

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

ANS:

```
package Day_7;

public class PassByValue {
   public static void changeValue(int x) {
      x = 20;
   }

   public static void main(String[] args) {
      int num = 10;
      System.out.println("Before: " + num);

      changeValue(num);

      System.out.println("After: " + num);
   }
}
```

Output:

```
Before: 10
After: 10
```

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

ANS:

```
package Day_7;

public class CallByReferenceSim {
   public static void changeArray(int[] arr) {
      arr[0] = 99;
   }
```

```
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3};

        System.out.println("Before: " + numbers[0]);
        changeArray(numbers);
        System.out.println("After: " + numbers[0]);
    }
}
```

Output:

```
Before: 1
After: 99
```

MultiThreading

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

ANS:

```
package Day_7;

public class ThreadExtendsDemo extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(i);
        }
    }

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

Output:

```
1
2
3
4
5
```

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

ANS:

```
package Day_7;

public class ThreadRunnableDemo implements Runnable {
    public void run() {
        System.out.println("Thread name: " + Thread.currentThread().getName());
    }

    public static void main(String[] args) {
        Thread t = new Thread(new ThreadRunnableDemo());
        t.start();
    }
}
```

Output:

```
Thread name: Thread-0
```

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

ANS:

```
package Day_7;

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

Output:

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

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

ANS:

```
package Day_7;

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

Output:

```
1
```

```
2
3
```
5 Create a thread and use Thread.yield() to pause and give chance to another thread.

ANS:

```
package Day_7;

public class YieldDemo extends Thread {
   public void run() {
      for (int i = 1; i <= 3; i++) {
         System.out.println(Thread.currentThread().getName() + ": " + i);
         Thread.yield();
      }
   }

   public static void main(String[] args) {
      new YieldDemo().start();
      new YieldDemo().start();
   }
}
```
Output:

```
Thread-0: 1
Thread-0: 2
Thread-1: 1
Thread-0: 3
Thread-1: 2
Thread-1: 3
```
6 Implement a program where two threads print even and odd numbers respectively.

ANS:

```
package Day_7;

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

```
Even: 2
Even: 4
Even: 6
Even: 8
Even: 10
Odd: 1
```

```
Odd: 3
Odd: 5
Odd: 7
Odd: 9
```

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

ANS:

```
package Day_7;

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

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

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

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

Output:

```
Thread-2 Priority: 10
Thread-1 Priority: 5
Thread-0 Priority: 1
```

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

ANS:

```
package Day_7;

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

    public static void main(String[] args) throws InterruptedException {
        JoinDemo t1 = new JoinDemo();
        t1.start();
        t1.join();
        System.out.println("Main thread ends after t1");
    }
}
```

```
Output:
Thread-0 1
Thread-0 2
Thread-0 3
Main thread ends after t1
```
9 Show how to stop a thread using a boolean flag.

ANS: package Day_7;

```
public class StopThreadDemo extends Thread {
   boolean running = true;

   public void run() {
      int i = 1;
      while (running) {
         System.out.println("Count: " + i++);
      }
   }

   public static void main(String[] args) throws InterruptedException {
      StopThreadDemo t = new StopThreadDemo();
      t.start();
      Thread.sleep(500);
      t.running = false;
   }
}
```
```
Output:
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Count: 6
Count: 7
Count: 8
Count: 9
Count: 10……………
Count: 6648
```

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

ANS:

```
package Day_7;

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

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

Output:

```
Count: 1905
```

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

ANS: package Day_7;

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

public class SyncDemo {
    public static void main(String[] args) throws InterruptedException {
        SyncCounter c = new SyncCounter();
        Thread t1 = new Thread(() -> { for (int i = 0; i < 1000; i++) c.increment(); });
        Thread t2 = new Thread(() -> { for (int i = 0; i < 1000; i++) c.increment(); });
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Count: " + c.count);
    }
}
```

Output:

```
Count: 2000
```

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

ANS:

```java
package Day_7;

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

public class SyncBlockDemo {
    public static void main(String[] args) throws InterruptedException {
        BlockCounter c = new BlockCounter();
        Thread t1 = new Thread(() -> { for (int i = 0; i < 1000; i++) c.increment(); });
        Thread t2 = new Thread(() -> { for (int i = 0; i < 1000; i++) c.increment(); });
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Count: " + c.count);
    }
}
```

Output:
Count: 2000

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

ANS:

```
package Day_7;

class BankAccount {
    int balance = 1000;
    synchronized void deposit(int amt) { balance += amt; }
    synchronized void withdraw(int amt) { balance -= amt; }
}

public class BankDemo {
    public static void main(String[] args) throws InterruptedException {
        BankAccount acc = new BankAccount();
        Thread t1 = new Thread(() -> acc.deposit(500));
        Thread t2 = new Thread(() -> acc.withdraw(200));
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println("Final Balance: " + acc.balance);
    }
}
```

Output:
Final Balance: 1300

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

ANS:

```
package Day_7;

class Buffer {
    int data;
    boolean available = false;

    synchronized void produce(int value) {
        try {
            while (available) wait();
            data = value;
            System.out.println("Produced: " + value);
            available = true;
            notify();
        } catch (Exception e) {}
    }

    synchronized void consume() {
        try {
            while (!available) wait();
            System.out.println("Consumed: " + data);
            available = false;
            notify();
        } catch (Exception e) {}
    }
}

public class ProducerConsumer {
    public static void main(String[] args) {
        Buffer b = new Buffer();
        new Thread(() -> { for (int i = 1; i <= 5; i++) b.produce(i); }).start();
        new Thread(() -> { for (int i = 1; i <= 5; i++) b.consume(); }).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.

ANS:

```
package Day_7;

class PrintTask {
    boolean letterTurn = true;
    synchronized void printLetter(char c) {
        try {
            while (!letterTurn) wait();
            System.out.print(c + " ");
            letterTurn = false;
            notify();
        } catch (Exception e) {}
    }
    synchronized void printNumber(int n) {
        try {
            while (letterTurn) wait();
            System.out.print(n + " ");
            letterTurn = true;
            notify();
        } catch (Exception e) {}
    }
}

public class AlternatePrint {
    public static void main(String[] args) {
        PrintTask task = new PrintTask();
        new Thread(() -> { for (char c = 'A'; c <= 'Z'; c++) task.printLetter(c); }).start();
        new Thread(() -> { for (int i = 1; i <= 26; i++) task.printNumber(i); }).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().

ANS:

```
package Day_7;

class SharedData {
    synchronized void display() {
        System.out.println(Thread.currentThread().getName() + " running");
        notifyAll();
    }
}

public class NotifyAllDemo {
    public static void main(String[] args) {
        SharedData s = new SharedData();
        Runnable r = () -> s.display();
        new Thread(r, "T1").start();
        new Thread(r, "T2").start();
        new Thread(r, "T3").start();
    }
}
```

Output:

```
T2 running
T3 running
T1 running
```

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

ANS:

```java
package Day_7;

public class DaemonDemo extends Thread {
   public void run() {
      while (true) {
         System.out.println("Daemon running...");
         try { Thread.sleep(1000); } catch (Exception e) {}
      }
   }

   public static void main(String[] args) {
      DaemonDemo d = new DaemonDemo();
      d.setDaemon(true);
      d.start();
      try { Thread.sleep(3000); } catch (Exception e) {}
      System.out.println("Main ends");
   }
}
```

Output:

```
Daemon running...
Daemon running...
Daemon running...
Main ends
```

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

ANS:

```java
package Day_7;

public class AliveDemo extends Thread {
   public void run() {
      System.out.println("Thread running");
   }

   public static void main(String[] args) throws InterruptedException {
      AliveDemo t = new AliveDemo();
      System.out.println("Before start: " + t.isAlive());
      t.start();
      System.out.println("After start: " + t.isAlive());
      t.join();
      System.out.println("After finish: " + t.isAlive());
   }
}
```

Output:

```
Before start: false
Thread running
After start: true
After finish: false
```

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

ANS:

```
package Day_7;

public class ThreadGroupDemo {
    public static void main(String[] args) {
        ThreadGroup group = new ThreadGroup("MyGroup");
        Runnable task = () -> System.out.println(Thread.currentThread().getName());
        new Thread(group, task, "T1").start();
        new Thread(group, task, "T2").start();
        System.out.println("Active threads in group: " + group.activeCount());
    }
}
```

Output:

```
T1
T2
Active threads in group: 2
```

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

ANS:

```
package Day_7;

import java.util.concurrent.*;

public class CallableDemo {
    public static void main(String[] args) throws Exception {
        Callable<Integer> task = () -> 5 * 4;
        ExecutorService service = Executors.newSingleThreadExecutor();
        Future<Integer> result = service.submit(task);
        System.out.println("Result: " + result.get());
        service.shutdown();
    }
}
```

Output:

```
Result: 20
```