Wrapper classes

1. Check if character is a Digit
2. Compare two Strings
3. Convert using valueof method
4. Create Boolean Wrapper usage
5. Convert null to wrapper classes

```
package day7_Assessments;
public class WrapperClasses{
    public static void main(String[] args) {
        // 1. Check if character is a Digit
        char ch = '5';
        System.out.println("Is '" + ch + "' a digit? " +
Character.isDigit(ch));
        // 2. Compare two Strings
        String str1 = "Hello";
        String str2 = "World";
        System.out.println("Comparing str1 and str2: " +
str1.equals(str2));
        System.out.println("Comparing str1 with 'Hello': " +
str1.equals("Hello"));
        // 3. Convert using valueOf method
        int num = 42;
        String numStr = String.valueOf(num);
        System.out.println("Converted int to String: " + numStr);
        double d = 3.14;
        Double dWrapper = Double.valueOf(d);
        System.out.println("Converted double to Double: " + dWrapper);
        // 4. Create Boolean Wrapper usage
        Boolean bool1 = Boolean.valueOf(true);
        Boolean bool2 = Boolean.valueOf("false");
        System.out.println("Boolean 1: " + bool1);
        System.out.println("Boolean 2: " + bool2);
        // 5. Convert null to wrapper classes
        String nullStr = null;
        try {
            Integer nullInt = Integer.valueOf(nullStr);
            System.out.println("Converted null to Integer: " + nullInt);
        } catch (NumberFormatException e) {
            System.out.println("Cannot convert null String to Integer: " +
e);
        }
        Integer wrapperNull = null;
        System.out.println("Wrapper directly assigned null: " +
wrapperNull);
    }
}
```

Output:

```
Is '5' a digit? true
```

```
Comparing str1 and str2: false
Comparing str1 with 'Hello': true
Converted int to String: 42
Converted double to Double: 3.14
Boolean 1: true
Boolean 2: false
Cannot convert null String to Integer: java.lang.NumberFormatException:
Cannot parse null string
Wrapper directly assigned null: null
```

Pass by value and pass by reference

1. Write a program where a method accepts an integer parameter and tries to change its value. Print the value before and after the method call.

```
package day7_Assessments;
public class PassByValue {
    public static void changeValue(int num) {
        System.out.println("Inside method before change: " + num);
        num = num + 10;
        System.out.println("Inside method after change: " + num);
    }
    public static void main(String[] args) {
        int original = 5;
        System.out.println("Before method call: " + original);
        changeValue(original);
        System.out.println("After method call: " + original);
    }
}
```

Output:

```
Before method call: 5
Inside method before change: 5
Inside method after change: 15
After method call: 5
```

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

```
package day7_Assessments;
public class SwapTwoNums {
    public static void swap(int a, int b) {
        System.out.println("Inside method before swap: a = " + a + ", b = "
+ b);
        int temp = a;
        a = b;
```

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

}
```

Output:

```
Before method call: x = 10, y = 20
Inside method before swap: a = 10, b = 20
Inside method after swap: a = 20, b = 10
After method call: x = 10, y = 20
```

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

```
package day7_Assessments;
public class PrimitivePass{

    public static void modifyValues(int a, double b, char c) {
        System.out.println("Inside method before change: a = " + a + ", b =
" + b + ", c = " + c);
        a = a + 10;
        b = b * 2;
        c = 'Z';
        System.out.println("Inside method after change: a = " + a + ", b =
" + b + ", c = " + c);
    }
    public static void main(String[] args) {
        int num = 5;
        double value = 3.5;
        char letter = 'A';
        System.out.println("Before method call: num = " + num + ", value =
" + value + ", letter = " + letter);
        modifyValues(num, value, letter);
        System.out.println("After method call: num = " + num + ", value = "
+ value + ", letter = " + letter);
    }
}
```

Output:

```
Before method call: num = 5, value = 3.5, letter = A
Inside method before change: a = 5, b = 3.5, c = A
Inside method after change: a = 15, b = 7.0, c = Z
```

```
After method call: num = 5, value = 3.5, letter = A
```

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.

```
package day7_Assessments;
class Box {
    int length;
    Box(int length) {
        this.length = length;
    }
    static void changeLength(Box b, int newLength) {
        b.length = newLength;
    }
}
public class BoxCallByValue {
    public static void main(String[] args) {
        Box myBox = new Box(10);
        System.out.println("Before change: " + myBox.length);
        Box.changeLength(myBox, 20);
        System.out.println("After change: " + myBox.length);
    }
}

output:
Before change: 10
After change: 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.

```
package day7_Assessments;
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```
    static void updatePerson(Person p, String newName, int newAge) {
        p.name = newName;
        p.age = newAge;
    }
}
public class PassByObject {
    public static void main(String[] args) {
        Person person = new Person("Alice", 25);
        System.out.println("Before update: " + person.name + ", " +
person.age);
        Person.updatePerson(person, "Bob", 30);
        System.out.println("After update: " + person.name + ", " +
person.age);
    }
}
```

Output:

```
Before update: Alice, 25
After update: Bob, 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.

```
package day7_Assessments;
class Student {
    String name;
    int marks;
    Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
    static void updateMarks(Student s, int newMarks) {
        s.marks = newMarks;
    }
}

public class StudentMarks {
    public static void main(String[] args) {
        Student student = new Student("John", 80);
        System.out.println("Before update: " + student.name + " - " +
student.marks);
        Student.updateMarks(student, 95);
        System.out.println("After update: " + student.name + " - " +
student.marks);
    }
}
```

Output:

```
Before update: John - 80
After update: John - 95
```

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

```
package day7_Assessments;
class Box1 {
    int length;

    Box1(int length) {
        this.length = length;
    }

    static void changeReference(Box1 b) {
        b = new Box1(50);
    }
}

public class CallByValue {
    public static void main(String[] args) {
        Box1 myBox = new Box1(10);
        System.out.println("Before changeReference: " + myBox.length);
        Box1.changeReference(myBox);
        System.out.println("After changeReference: " + myBox.length);
    }
}
```

Output:

```
Before changeReference: 10
After changeReference: 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.

```
package day7_Assessments;
class Car {
    String model;

    Car(String model) {
        this.model = model;
    }
    static void assignNewObject(Car c) {
        c = new Car("Tesla");
    }
}
public class CarReference {
    public static void main(String[] args) {
        Car myCar = new Car("BMW");
        System.out.println("Before assignNewObject: " + myCar.model);
```

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

Output:

```
Before assignNewObject: BMW
After assignNewObject: BMW
```

---

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

```
package day7_Assessments;
public class PrimitiveNonPrimitive {
    static void changeValue(int x) {
        x = 50;
    }
    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.

```
package day7_Assessments;
class IntWrapper {
    int value;
    IntWrapper(int value) {
        this.value = value;
    }
}
public class CallByReferenceSimulation {
    static void modify(IntWrapper num) {
        num.value = num.value + 10;
    }
    public static void main(String[] args) {
```

```
        IntWrapper number = new IntWrapper(5);
        System.out.println("Before: " + number.value);
        modify(number);
        System.out.println("After: " + number.value);
    }
}
```

Output:

```
Before: 5
After: 15
```

MultiThreading

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

```
package day7_Assessments;
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) {
                System.out.println("Thread interrupted");
            }
        }
    }
}
public class ThreadExample1 {
    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.

```
package day7_Assessments;
class MyRunnable implements Runnable {
```

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

Output:

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

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

```
package day7_Assessments;
class MessageThread extends Thread {
    private String message;
    public MessageThread(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 TwoThreadsExample {
    public static void main(String[] args) {
        MessageThread t1 = new MessageThread("Hello from Thread 1");
        MessageThread t2 = new MessageThread("Greetings from Thread 2");
        t1.start();
        t2.start();
    }
}
```

Output:

```
Hello from Thread 1 - 1
Greetings from Thread 2 - 1
Hello from Thread 1 - 2
Greetings from Thread 2 - 2
Hello from Thread 1 - 3
Greetings from Thread 2 - 3
Hello from Thread 1 - 4
```

```
Greetings from Thread 2 - 4
Hello from Thread 1 - 5
Greetings from Thread 2 - 5
```

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

```
package day7_Assessments;
public class SleepExample {
    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.

```
package day7_Assessments;
class YieldThread extends Thread {
    public YieldThread(String name) {
        super(name);
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(getName() + " - Count: " + i);
            if (i == 3) {
                System.out.println(getName() + " is yielding...");
                Thread.yield();
            }
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                System.out.println(getName() + " interrupted");
            }
        }
    }
}
public class ThreadYieldExample {
    public static void main(String[] args) {
        YieldThread t1 = new YieldThread("Thread-A");
        YieldThread t2 = new YieldThread("Thread-B");

        t1.start();
```

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

Output:

```
Thread-B - Count: 1
Thread-A - Count: 1
Thread-B - Count: 2
Thread-A - Count: 2
Thread-B - Count: 3
Thread-B is yielding...
Thread-A - Count: 3
Thread-A is yielding...
Thread-B - Count: 4
Thread-A - Count: 4
Thread-B - Count: 5
Thread-A - Count: 5
```

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

```
package day7_Assessments;
class EvenThread extends Thread {
    private final int max;
    public EvenThread(int max) {
        this.max = max;
    }
    public void run() {
        for (int i = 2; i <= max; i += 2) {
            System.out.println("Even Thread: " + i);
            try {
                Thread.sleep(200);  // Pause for visibility
            } catch (InterruptedException e) {
                System.out.println("Even Thread interrupted");
            }
        }
    }
}
class OddThread extends Thread {
    private final int max;

    public OddThread(int max) {
        this.max = max;
    }
    public void run() {
        for (int i = 1; i <= max; i += 2) {
            System.out.println("Odd Thread: " + i);
            try {
                Thread.sleep(200);  // Pause for visibility
            } catch (InterruptedException e) {
                System.out.println("Odd Thread interrupted");
            }
        }
    }
}
```

```
public class EvenOddThreads {
    public static void main(String[] args) {
        int maxNumber = 10;
        EvenThread evenThread = new EvenThread(maxNumber);
        OddThread oddThread = new OddThread(maxNumber);
        evenThread.start();
        oddThread.start();
    }
}
```

Output:

```
Even Thread: 2
Odd Thread: 1
Even Thread: 4
Odd Thread: 3
Even Thread: 6
Odd Thread: 5
Odd Thread: 7
Even Thread: 8
Odd Thread: 9
Even Thread: 10
```