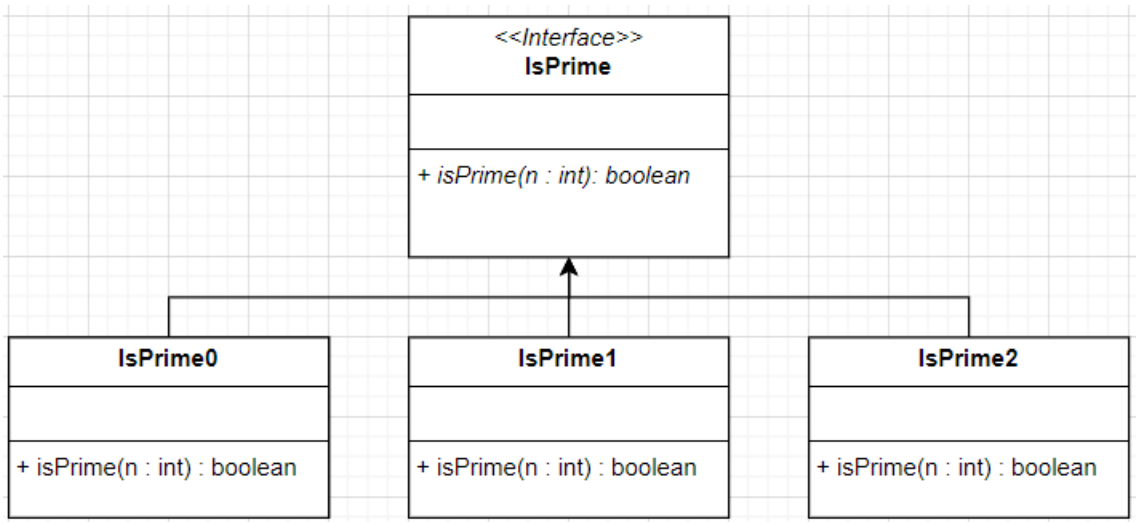
Objective(s):

- To practice on analyzing algorithms' runtime

Task 1: Implement IsPrime0, IsPrime1 and IsPrim2. (in .\solutions\pack2)




```java
package solutions.pack2;

public interface L2_IsPrimeInterface {
    boolean isPrime(int n);
}
```

```java
// IsPrime0
public boolean isPrime(int n) {
  if (n == 1) return false;
  if (n <= 3) return true;
  int m = n/2;
  for (int i = 2; i <= m; i++) {
    if (n % i == 0) return false;
  }
  return true;
}
```

```java
// IsPrime1
public boolean isPrime(int n) {
  if (n == 1) return false;
  if (n <= 3) return true;
  int m = (int)Math.sqrt(n);
  for (int i = 2; i <= m; i++) {
      if (n % i == 0) return false;
  }
  return true;
}
```

The method isPrime0(n) takes any positive integer and returns true if it is a prime, false otherwise. The method run through all integer from 2 to n/2 and check if n is divisible by any of them.

There are two more methods, isPrime1(n) and isPrime2(n). The method isPrime1(n) is similar to isPrime0(n) but only run from 2 to $\sqrt{n}$. The method isPrime2(n) improves upon isPrime1(n) by take out anything divisible by 2 and 3 and not going to test divisibility of number that are multiple of 2 and 3.

For testing, we can use the following program:

```
private static void testIsPrime012() {
    int N = 100;
    int count = 0;
    L2_IsPrimeInterface obj = new IsPrime0();
    for (int n = 1; n < N; n++) {
        if (obj.isPrime(n)) count++;
    }
    System.out.println("Pi ("+ N + ")= " + count);

    count = 0;
    obj = new IsPrime1();
    for (int n = 1; n < N; n++) {
        if (obj.isPrime(n)) count++;
    }
    System.out.println("Pi ("+ N + ")= " + count);

    count = 0;
    obj = new IsPrime2();
    for (int n = 1; n < N; n++) {
        if (obj.isPrime(n)) count++;
    }
    System.out.println("Pi ("+ N + ")= " + count);
}
```

Remark : There are 25 prime numbers between 2 to 100.

Task 2: run the program with isPrime0, isPrime1, and isPrime2. Record your result into the following table.

Running-time table					
n	numPrime(n)	time (milliseconds)			
		Lab's isPrime0	isPrime0	isPrime1	isPrime2
100,000		353			
200,000		1,283			
300,000		2,792			
400,000		4,820			
500,000		7,370			
600,000		15,580			
700,000		24,557			
800,000		31,716			
900,000		39,964			
1,000,000		48,785			

```
public static void bench_isPrime(IsPrimeInterface obj) {
  int your_cpu_factor = 1; /* increase by 10 times */
  int N = 100;
  int count = 0;
        // long start = 0;
  for (N = 100_000; N <= 1_000_000 * your_cpu_factor; N+= 100_000 * your_cpu_factor) {
      long start = System.currentTimeMillis();
      for (int n = 1; n < N; n++) {
            if (obj.isPrime(n)) count++;
      }
      long time = (System.currentTimeMillis() - start);
      System.out.println(N + "\t" + count + "\t" + time);
  }
}
```

Taks 3 : Analyze whether time increased on isPrime0 is linear.

Running-Time Analysis							
n	Data size ratio	Lab's isPrime0	Time increased(%)	Time increased factor	your isPrime0	Time increased(%)	Time increased factor
100,000	n	353	1.00000				
200,000	2n	1,283	3.63456	3.63456			
300,000	3n	2,792	7.90935	2.17615			
400,000	4n	4,820	13.65439	1.72636			
500,000	5n	7,370	20.44135	1.52905			
600,000	6n	15,580	44.13598	2.11398			
700,000	7n	24,557	69.56657	1.57619			
800,000	8n	31,716	89.84703	1.29153			
900,000	9n	39,964	113.21246	1.26006			
1,000,000	10n	48,785	138.20113	1.22072			

Task 4: Plot runtime graph your isPrime0's vs. your isPrime1's and isPrime2's

Submission: this pdf.

Due Date: TBA