1. Given the binary search tree you saw in class,

```java
public class BinarySearchTree<AnyType extends Comparable<AnyType>> {
    //...
    // Basic node stored in unbalanced binary search trees
    private static class BinaryNode<AnyType> {
        //...
        AnyType element; // The data in the node
        BinaryNode<AnyType> left; // Left child
        BinaryNode<AnyType> right; // Right child
    }
    /** The tree root. */
    private BinaryNode<AnyType> root;
}
```

write a method with the following signature:

```java
LinkedList findInRange(Anytype xmin, Anytype xmax),
```

that returns a list of elements in the tree within the \([xmin, xmax]\) range. Your method should explore the fact that it is a binary search tree and cut computations whenever possible.
2. An algebraic expression is said to be in a sum of products form if it a sum of products of variables. For instance,

\[ x \times y + z \times w \]
\[ a + b \times c + d \times e \times f \]

are sum of products, while

\[ (x + y) \times (z + w) \]
\[ a \times (b + c) \]

are not.

In this exercise you’ll be given a string representing an algebraic expression in postfix notation (for simplicity) with only additions and multiplications, and you’re supposed to print an equivalent sum of products expression in infix notation.

Sample input: \( x \ y + z \ w + * \) \hspace{1cm} (infix: \( (x + y) \times (z + w) \))
Sample output: \( x \times z + x \times w + y \times z + y \times w \)

Use whatever data structures you find convenient.
3. Given the `DoubledLinkedList` class you saw in class:

```java
class DoubleListNode
{
    // Constructors
    DoubleListNode( Object theElement )
    {
        this( theElement, null, null );
    }

    DoubleListNode( Object theElement, DoubleListNode previous, DoubleListNode next )
    {
        element = theElement;
        this.previous = previous;
        this.next = next;
    }

    // Friendly data; accessible by other package routines
    Object element;
    DoubleListNode previous;
    DoubleListNode next;
}

public class DoubleLinkedList
{
    private DoubleListNode header;
    private DoubleListNode tailer;

    public DoubleLinkedList( )
    {
        header = new DoubleListNode ( null );
        tailer = new DoubleListNode ( null, header, null);
        header.next = tailer;
    }
}
```

(a) Write the following method that appends a doubled linked list to another:

```java
public void append(DoubleLinkedList L2)
```

(b) Write a method with the following signature that shifts the list circularly to the left making `p` the new header:

```java
public void leftCircularShift(DoubleLinkedListItr p)
```

For instance, given the list \((a, b, c, d, e)\), a circular shift to the left at `c` would transform the list into \((c, d, e, a, b)\). Your algorithm must run in constant time.
4. In this exercise you are going to simulate customers in a line.

The following information is given about each customer:

- His/her arrival time
- The amount of work he/she requires from the cashier
- The time he/she is willing to wait in line before getting bored and leaving

So a customer could be represented by the following class:

```java
public class Customer
{
    String name;
    public double arrival;
    public double work;
    public double tolerance;

    Customer(String n, double a, double w, double t)
    {
        name = n;
        arrival = a;
        work = w;
        tolerance = t;
    }
}
```

The cashier is described simply by his speed $s$, that is, he serves a customer with an amount of work $w$ in $w/s$ minutes. So we represent a cashier by the class:

```java
public class Cashier
{
    String name;
    double speed;

    Cashier(String n, double s)
    {
        name = n;
        speed = s;
    }
}
```

(Feel free to modify the Cashier and Customer classes if necessary.)

(a) First assume that there’s only one cashier working and you are given a list of customers. Your task is to print the time that each customer waited in line, or that he got bored and left. More precisely, you have to write a method with the following signature:

```java
public void simulateLine(Customer[] customers, Cashier cashier);
```

For instance, for the input
Customer[] customers = new Customer[3];
customer[0] = new Customer("John", 0.0, 10.0, 2.0);
customer[1] = new Customer("Martha", 2.5, 5.0, 5.0);
customer[2] = new Customer("Bill", 2.0, 3.0, 1.0);

Cashier cashier = new Cashier("Ed", 2.0);

simulateLine(customers, cashier);

you should print:

John waited 0 minutes in line (served by Ed)
Bill got bored and left
Martha waited 2.5 minutes in line (served by Ed)

(b) Now assume that there’s more than one cashier working. Once a cashier is done assisting a
customer he immediately gets the next customer in line (in case multiple cashiers are free at the
same time, the next customer can go to any of them).
The new method should have the signature:

public void simulateLine(Customer[] customers, Cashier[] cashiers);

And the output should follow the format of item a.

For instance, for the input

Customer[] customers = new Customer[4];
customers[0] = new Customer("John", 0.0, 10.0, 2.0);
customers[1] = new Customer("Martha", 2.5, 5.0, 5.0);
customers[2] = new Customer("Bill", 2.0, 3.0, 1.0);
customers[3] = new Customer("Henry", 2.75, 2.0, 3.0);

Cashier cashiers[] = new Cashier[2];
cashiers[0] = new Cashier("Ed", 2.0);
cashiers[1] = new Cashier("Paul", 4.0);

simulateLine(customers, cashiers);

you should print

John waited 0.0 minutes in line (served by Ed)
Bill waited 0.0 minutes in line (served by Paul)
Martha waited 0.25 minutes in line (served by Paul)
Henry waited 1.25 minutes in line (served by Paul)