Zadanie: dodatkowe operacje na listach

Stworzyć klasę XList, dostarczającą dodatkowych możliwości tworzenia list i operowania na nich.
W klasie powinny znaleźć się odpowiednie konstruktory oraz statyczne metody ***of***, umożliwiające tworzenie obiektów XList z innych kolekcji, tablic oraz argumentów podawanych przez przecinki.
Dodatkowo pomocnicze metody do tworzenia xlist z napisów:

- ofChars(napis) - zwraca x-listę znaków napisu,
- ofTokens(napis, [sep]) - zwraca x-listę symboli napisu, rozdzielonych separatorami z sep (jeśli brak - to białymi znakami).

Oprócz tego dostarczyć metod:

- union(dowolna_kolekcja) - zwraca nową x-list z dołączaną do tej x-list zawartością kolekcji,
- diff(dowolna_kolekcja) - zwraca x-list zawierającą te elementy tej x-list, które nie występują w kolekcji,
- unique() - zwraca nową x-list, która zawiera wszystkie niepowtarzające się elementy tej x-list
- combine() - zwraca x-listę list-kombinacji elementów z poszczególnych kolekcji, będących elementami tej x-listy
- collect(Function) - zwraca nową x-listę, której elemenatmi są wyniki funkcji stosowanej wobec elementów tej x-listy,
- join([sep]) - zwraca napis, będący połączeniem elementów tej x-listy, z ewentualnie wstawionym pomiędzy nie separatorem,
- forEachWithIndex(konsumer_z_dwoma argumentami: element, index) - do iterowania po liście z dostępem i do elementów i do ich indeksów.

Za realizację każdej z w/w właściwosci uzyskuje się odrębne punkty, tak że można wykonac tylko część zadania.
Przy tym należy jednak uważać, aby przekazany w rozwiązaniu plik Main.java nie miał błędów w kompilacji.

Klasa Main zawarta w projekcie powinna dobrze wyjaśniać sposób realizacji zadania:

```
import java.util.*;

// Plik Main.java może być dowolnie modyfikowany,
// ale punkty uzyskuje się za właściwe dzialanie poszczególnych pokazanych tu metod klasy XList.

// Jeżeli nie oprogramujemy wszystkich metod, to z klasy Main nalezy usunąć te fragmenty,
// które powodują błędy w kompilacji - w przeciwnym razie nie uzyskamy punktów.

public class Main {
  public static void main(String[] args) {
    // Pewne dodatkowe zestawy danych
    Integer[] ints = { 100, 200, 300 };
    Set<Integer> set = new HashSet<>(Arrays.asList(3, 4, 5));

    // Sposoby tworzenia
    XList<Integer> list1 = new XList<>(1, 3, 9, 11);
    XList<Integer> list2 = XList.of(5, 6, 9);
    XList<Integer> list3 = new XList(ints);
    XList<Integer> list4 = XList.of(ints);
    XList<Integer> list5 = new XList(set);
    XList<Integer> list6 = XList.of(set);

    System.out.println(list1);
    System.out.println(list2);
    System.out.println(list3);
    System.out.println(list4);
    System.out.println(list5);
    System.out.println(list6);

    // --- i pomocnicze metody do tworzenia z napisów
    XList<String> slist1 = XList.charsOf("ala ma kota");
    XList<String> slist2 = XList.tokensOf("ala ma kota");
    XList<String> slist3 = XList.tokensOf("A-B-C", "-");

    System.out.println(slist1);
    System.out.println(slist2);
    System.out.println(slist3);

    // Metoda union - suma elementów
    List<Integer> m1 = list1.union(list2);  // oczywiście, można podstawiać na List
    System.out.println(m1);
    // można wykonywać wszystkie operacje z interfejsu List, np:
    m1.add(11);
    System.out.println(m1);
    XList<Integer> m2 = (XList<Integer>) m1;
    XList<Integer> m3 = m2.union(ints).union(XList.of(4, 4));
    System.out.println(m2); // m2 się nie zmienia
    System.out.println(m3); // wynik jest w m3
    m3 = m3.union(set);
    System.out.println(m3);

    // Widzieliśmy metode union
    // Teraz metoda diff(dowolna kolekcja)
    System.out.println(m3.diff(set));  // wszystko z m3, co nie jest w set
    System.out.println(XList.of(set).diff(m3)); // co jest w set, czego nie ma w m3

    // Metoda unique -zwraca nową Xlist bez duplikatow
    XList<Integer> uniq = m3.unique(); // lista, nie Set
    System.out.println(uniq);

    // kombinacje (kolejność jest istotna)
    List<String> sa = Arrays.asList( "a", "b");
    List<String> sb = Arrays.asList( "X", "Y", "Z" );
    XList<String> sc = XList.charsOf( "12" );
    XList toCombine = XList.of(sa, sb, sc);  // czy można tu uniknąć użycia typu surowego?
    System.out.println(toCombine);
    XList<XList<String>> cres = toCombine.combine();
    System.out.println(cres);

    // collect i join
    XList<String> j1 = cres.collect( list -> list.join());
    System.out.println(j1.join(" "));
    XList<String> j2 =cres.collect( list -> list.join("-"));
    System.out.println(j2.join(" "));

    // forEachWithIndex
    XList<Integer> lmod = XList.of(1,2,8, 10, 11, 30, 3, 4);
    lmod.forEachWithIndex( (e, i) -> lmod.set(i, e*2));
    System.out.println(lmod);
    lmod.forEachWithIndex( (e, i) -> { if (i % 2 == 0) lmod.remove(e); } );
    System.out.println(lmod);
    lmod.forEachWithIndex( (e, i) -> { if (i % 2 == 0) lmod.remove(i); } );
    System.out.println(lmod); // Pytanie: dlaczego mamy taki efekt?

  }
}
```

Ten program wypisuje na konsoli:

```
[1, 3, 9, 11]
[5, 6, 9]
[100, 200, 300]
[100, 200, 300]
[3, 4, 5]
[3, 4, 5]
[a, l, a,  , m, a,  , k, o, t, a]
[ala, ma, kota]
[A, B, C]
[1, 3, 9, 11, 5, 6, 9]
[1, 3, 9, 11, 5, 6, 9, 11]
[1, 3, 9, 11, 5, 6, 9, 11]
[1, 3, 9, 11, 5, 6, 9, 11, 100, 200, 300, 4, 4]
[1, 3, 9, 11, 5, 6, 9, 11, 100, 200, 300, 4, 4, 3, 4, 5]
[1, 9, 11, 6, 9, 11, 100, 200, 300]
[]
[1, 3, 9, 11, 5, 6, 100, 200, 300, 4]
[[a, b], [X, Y, Z], [1, 2]]
[[a, X, 1], [b, X, 1], [a, Y, 1], [b, Y, 1], [a, Z, 1], [b, Z, 1], [a, X, 2], [b, X, 2], [a, Y, 2], [b, Y, 2], [a, Z, 2], [b, Z, 2]]
aX1 bX1 aY1 bY1 aZ1 bZ1 aX2 bX2 aY2 bY2 aZ2 bZ2
a-X-1 b-X-1 a-Y-1 b-Y-1 a-Z-1 b-Z-1 a-X-2 b-X-2 a-Y-2 b-Y-2 a-Z-2 b-Z-2
[2, 4, 16, 20, 22, 60, 6, 8]
[4, 16, 22, 60, 8]
[16, 22, 60, 8]
```