Zadanie A - Książka telefoniczna

Zaimplementuj książkę telefoniczną - klasę przechowującą numery telefonów, z możliwością ich grupowania, według poniższego schematu:

```
public class PhoneBook {
    public enum NumberFormat{ DIGITS, HYPHENED }

    public PhoneBook(){...}
    public PhoneBook(NumberFormat format){...}
    public PhoneBook(int capacity){...}
    public PhoneBook(NumberFormat format, int capacity){...}
    public PhoneBook copyBook(){...}

    public int size(){...}
    public int capacity(){...}
    public boolean isEmpty(){...}
    public boolean isFull(){...}

    public PhoneBook add(String number){...}
    public PhoneBook add(PhoneBook subBook){...}
    public void changeFormat(NumberFormat format){...}

    public boolean contains(String number){...}

    public boolean contains(PhoneBook pb){...}
    public boolean elementOf(PhoneBook pb){...}
    public boolean subsetOf(PhoneBook pb){...}
    public boolean supersetOf(PhoneBook pb){...}
    public boolean equals(Object o){...}

    public String toString(){...}
}
```

Klasę należy zaimplementować w pakiecie domyślnym, korzystając z podstawowych klas `String, StringBuilder, Object`. Nie można używać kontenetów (m.in. list) ze standardowej biblioteki. W szczególności: `javap -v PhoneBook.class | grep util` ma być pusty.

Klasa ma działać w następujący sposób

Tworzenie książki:

- `new PhoneBook(format,capacity)` tworzy pustą książkę z ustawionym formatem numerów i pojemnością
  - format opisuje w jakiej formie przyjmujemy i przechowujemy numery
    - DIGITS - 9 cyfr, *111111111*,
    - HYPHENED - 3 razy po 3 cyfry oddzielone '-', *111-111-111*
  - domyślny format to DIGITS, domyślne capacity to 10
- `copyBook()` tworzy kopię książki, niezależną od oryginału

Sprawdzanie stanu:

- `size()` zwraca liczbę wszystkich numerów w książce
- `capacity()` zwraca pojemność danej książki
- `isEmpty()` czy ksiażka jest pusta (nie zawiera żadnych numerów, ani żadnych podgrup)
- `isFull()` czy pojemność została wyczerpana

Zmiana zawartosci:

- `add(number)` dodaje numer (chyba, że została dodana już jakaś podksiążka - wtedy brak efektu); numer zostaje dodany jeśli jest w poprawnym formacie
- `add(subBook)` dodaje podksiążkę o takiej zawartości i strukturze jak podana książka (chyba, że były już dodane wcześniej numery - wtedy brak efektu), przy czym format jest dostosowywany do formatu książki do której dodajemy
- `changeFormat(format)` zmienia formatowanie numerów w książce

Dana książka może zawierać albo listę numerów (ograniczoną przez capacity) albo podksiążki (co najwyżej 10). Dodawanie działa jak operacja na zbiorach - jeśli dodawany element (podzbiór) już występował w zbiorze, nie dodajemy. Rozmiar książki to liczba przechowywanych numerów. W przypadku zbioru podksiążek jest to suma rozmiarów podksiążek. Rozmiar nie może przekroczyć pojemności. Funkcje `add()` zwracają referencję do obiektu na którym zostały wywołane.

Sprawdzanie zawartości:

- `contains(String number)` czy książka (lub podksiążka) zawiera numer (format ma znaczenie)

Dalsze operacje jak na zbiorach. Format nie ma znaczenia.

- `contains(PhoneBook pb)` czy książka zawiera podksiążkę jako element
- `elementOf(PhoneBook pb)` czy książka jest elementem podanej książki
- `subsetOf(PhoneBook pb)` czy książka jest podzbiorem podanej książki
- `supersetOf(PhoneBook pb)` czy książka jest nadzbiorem podanej książki
- `equals(Object o)` czy książki zawierają te same elementy/podksiążki (kolejność bez znaczenia)

Wypisanie:

- `toString()` wypisz reprezentację książki, według kolejności w jakiej były dodawane
  - w pierwszej linii `"{"`
  - w kolejnych liniach kolejne numery (poprzedzone wcięciem)
  - jeśli książka zawiera podksiążki, to rekurencyjnie reprezentacja podksiążki (z odpowiednio powiększonym wcięciem)
  - w ostatniej linii `"}"` (zakończone nową linią).

Przykładowe testy

```
@Test
public void testEmpty(){
    PhoneBook pb = new PhoneBook();
    assertTrue(pb.isEmpty());
    assertFalse(pb.isFull());
    assertEquals(0, pb.size());
    assertEquals(10, pb.capacity());
    PhoneBook empty = new PhoneBook();
    assertFalse(pb.contains(empty));
    assertTrue(pb.supersetOf(empty));
    assertTrue(pb.subsetOf(empty));
    assertTrue(pb.equals(empty));
    assertEquals("{\n}\n", pb.toString());
}
```

```
@Test
public void testAdd1() {
    PhoneBook pb = new PhoneBook(PhoneBook.NumberFormat.HYPHENED);
    pb.add("111");
    pb.add("11-22-33");
    pb.add("111-22-33");
    pb.add("12-222-33-33");
    pb.add("-222-333-333");
    pb.add("222-333-333-");
    assertTrue(pb.isEmpty());
    pb.add("222-333-333");
    pb.add("222444555");
    assertFalse(pb.isEmpty());
    assertEquals(1, pb.size());
}
@Test
public void testAdd2() {
    PhoneBook pb = new PhoneBook(PhoneBook.NumberFormat.DIGITS);
    pb.add("11100011");
    pb.add("111*00011");
    pb.add("111.223.233");
    pb.add("12-222-33-33");
    pb.add("-222-333-333");
    pb.add("222-333-333-");
    assertTrue(pb.isEmpty());
    pb.add("222-333-333");
    pb.add("+222444557");
    pb.add("222444556");
    pb.add("0222444555");
    pb.add("1222444555");
    assertFalse(pb.isEmpty());
    assertEquals(1,pb.size());
}
```

```
@Test
public void testSwitchFormat1() {
    PhoneBook pb = new PhoneBook(PhoneBook.NumberFormat.HYPHENED).add("111-222-333");
    assertFalse(pb.contains("111222333"));
    pb.changeFormat(PhoneBook.NumberFormat.DIGITS);
    assertTrue(pb.contains("111222333"));
}
@Test
public void testSwitchFormat2() {
    PhoneBook g1 = new PhoneBook(PhoneBook.NumberFormat.DIGITS).add("111222333");
    PhoneBook g2 = new PhoneBook(PhoneBook.NumberFormat.HYPHENED).add("111-222-444");
    PhoneBook pb = new PhoneBook(PhoneBook.NumberFormat.DIGITS).add(g1);
    assertTrue(pb.contains("111222333"));
    assertTrue(g2.contains("111-222-444"));
    assertFalse(pb.contains("111222444"));
    pb.add(g2);
    assertFalse(pb.contains("111-222-444"));
    assertTrue(pb.contains("111222444"));
}


@Test
public void testGrouping() {
    PhoneBook g0 = new PhoneBook().add("111000001").add("111000003");
    PhoneBook g1 = new PhoneBook().add("111000001").add("111000002").add("111000003");
    PhoneBook g2 = new PhoneBook().add("222000001").add("222000002").add("222000003");
    PhoneBook pb1 = new PhoneBook(7).add(g1).add(g2);
    PhoneBook pb2 = new PhoneBook().add(g2).add(g1);
    assertFalse(pb1.toString().equals(pb2.toString()));
    assertTrue(pb1.equals(pb2));
    assertEquals(6,pb1.size());
    assertEquals("{\n  {\n    111000001\n    111000002\n    111000003\n  }\n"
                 + "  {\n    222000001\n    222000002\n    222000003\n  }\n}\n",
                pb1.toString());
    assertFalse(pb1.contains(g0));
    assertFalse(pb1.supersetOf(g0));
    assertTrue(pb1.supersetOf(new PhoneBook().add(g1)));
    assertTrue(g1.supersetOf(g0));
    assertFalse(g1.contains(g0));
    assertTrue(pb1.contains("111000001"));
    assertTrue(pb1.contains("222000001"));
    assertFalse(pb1.contains("121000003"));
    PhoneBook revHyph = new PhoneBook(PhoneBook.NumberFormat.HYPHENED);
    revHyph.add("111-000-003").add("111-000-002").add("111-000-001");
    assertTrue(pb1.contains(revHyph));
    pb1.add(g0);
    pb2.add(g0);
    assertFalse(pb1.equals(pb2));
}
```

```java
@Test
public void testCopy() {
    PhoneBook g1 = new PhoneBook().add("111000001").add("111000002").add("111000003");
    PhoneBook g2 = g1.copyBook();
    g2.changeFormat(PhoneBook.NumberFormat.HYPHENED);
    assertEquals(3,g1.size());
    assertTrue(g1.equals(g2));
    g2.add("222-000-001");
    assertEquals(3,g1.size());
    assertEquals(4,g2.size());
    assertFalse(g1.equals(g2));
    assertTrue(g1.subsetOf(g2));
}
```