Zadanie C - Lista jest Funkcją

Każda lista jest funkcją. Na przykład lista `[Ala, ma, kota]` może być rozumiana jako funkcja: `{0=Ala, 1=ma, 2=kota}`. Należy zaimplementować klasę:

```
public class ListIsFunction {
    public List asList() { ... }
    public Map asMap() { ... }
}
```

która pozwala na dostęp **do tych samych danych** z jednej strony jako do listy Object'ów, a z drugiej jako do funkcji typu Object w Object (przy czym argument jest tak naprawdę zawsze typu Integer).

Metoda `asList()` powinna zwracać referencję udostępniającą nasze dane jako listę (obiekt implementujący interfejs `java.util.List`), a metoda `asMap()` zwracać referencję prezentującą nasze dane jako mapę (obiekt implementujący interfejs `java.util.Map`). Wszystkie zwrócone referencje dają dostęp **do tych samych danych** przechowywanych w obiekcie `ListIsFunction`, tj. zmiany na jednej referencji (np. na widoku listy) wpływają na wszystkie pozostałe istniejące (i przyszłe) referencje (w szczególności na widok mapy).

Konstruktor domyślny `ListIsFunction` tworzy pusty obiekt - bez żadnych danych. Zawartość można przeglądać i modyfikować operacjami jakie dostarcza interfejs List czy Map, zgodnie z ich specyfikacją. W przypadku funkcji (mapy) nie są dozwolone operacje, które spowodowałyby rozspójnienie danych (np. usunięcie z dziedziny funkcji numeru indeksującego element ze środka listy, czy ustawienie funkcji dla argumentu większego niż długość listy) - takie operacje powinny rzucać wyjątki (zgodnie ze specyfikacją interfejsu Map).

Rozwiązanie ma spełniać następujące warunki:

- należy dostarczyć plik `ListIsFunction.java`, implementujący klasę `ListIsFunction`
- klasa dziedziczy tylko po klasie `Object`
- **wszystkie pozostałe klasy** powinny być zaimplementowane jako **klasy anonimowe** tzn. bez użycia słowa `class`

Dodatkowe informacje:

- przeglądanie i wypisywanie elementów przez mapę powinno być w takiej samej kolejności jak dostęp w postaci listy
- jeśli podczas iterowania po liście/mapie obiekt zmieni swoją zawartość, zachowanie jest niewyspecyfikowane
- dokumentacja Javy opisuje List oraz Map jako typy parametryzowane; należy to zignorować i zaipmlementować je ignorując parametry typu (np. `List list = new ArrayList()`) i zakładając że wszystkie przechowywane byty są typu `Object`
- w razie trudności dokumentacja Javy może pomóc w implementacji zadanej funkcjonalności

Przykładowe testy

```
@Test
public void testBasic() {
    ListIsFunction temp = new ListIsFunction();
    assertEquals(Object.class, temp.getClass().getSuperclass());
    List tempList = temp.asList();
    Map tempMap = temp.asMap();
    assertEquals("[]", tempList.toString());
    assertEquals("{}",tempMap.toString());
    tempList.add("Ala");
    tempList.add("ma");
    tempList.add("kota");

    assertEquals("{0=Ala, 1=ma, 2=kota}",tempMap.toString());
    assertEquals("[0, 1, 2]",tempMap.keySet().toString());
    assertEquals("[Ala, ma, kota]",tempMap.values().toString());
    tempMap.put(1, "miala");
    assertEquals("[Ala, miala, kota]",tempList.toString());
    tempMap.put(3,"malego");
    assertEquals("[Ala, miala, kota, malego]",tempList.toString());
    tempMap.remove(3);
    assertEquals("[Ala, miala, kota]",tempList.toString());

    Iterator it = tempMap.keySet().iterator();
    assertEquals("Ala",tempMap.get(it.next()));
    assertEquals("miala",tempMap.get(it.next()));
    try {
        it.remove();
        fail("No exception!");
    } catch(Exception e) {
        assertEquals(java.lang.IllegalStateException.class, e.getClass());
    }
    assertEquals("kota",tempMap.get(it.next()));
    it.remove();
    assertEquals("[Ala, miala]", tempList.toString());

    try {
        tempMap.remove(0);
        fail("No exception!");
    } catch(Exception e) {
        assertEquals(java.lang.IllegalArgumentException.class, e.getClass());
    }
    tempMap.remove(1);
    tempMap.remove(0);
    assertEquals("[]", tempList.toString());
    assertTrue(tempList.isEmpty());
    assertTrue(tempMap.isEmpty());
}
```