Lock-free контейнер

> GitHub-classroom для самостоятельно изучающих курс **не** в рамках университетских программ

Необходимо реализовать в lock-free стиле следующий интерфейс:

```
/**
 * Lock-Free множество.
 * @param <T> Тип ключей
 */
public interface Set<T extends Comparable<T>> {
    /**
   * Добавить ключ к множеству
     *
   * Алгоритм должен быть как минимум lock-free
     *
     * @param value значение ключа
     * @return false если value уже существует в множестве, true если элемент был добавлен
     */
    boolean add(T value);


    /**
   * Удалить ключ из множества
     *
   * Алгоритм должен быть как минимум lock-free
     *
     * @param value значение ключа
     * @return false если ключ не был найден, true если ключ успешно удален
     */
    boolean remove(T value);


    /**
   * Проверка наличия ключа в множестве
     *
   * Алгоритм должен быть как минимум wait-free для типов конечной размерноости и lock-free для остальных
     *
     * @param value значение ключа
     * @return true если элемент содержится в множестве, иначе - false
     */
    boolean contains(T value);


    /**
   * Проверка множества на пустоту
     *
```

```
    * Алгоритм должен быть как минимум lock-free
     *
     * @return true если множество пусто, иначе - false
     */
    boolean isEmpty();

    /**
    * Возвращает lock-free итератор для множества
     *
    * Итератор должен быть линеаризуем в терминах представления когда-либо существовавшего вместе набора элементов
     *
     * @return итератор для множества
     */
    java.util.Iterator<T> iterator();
}
```

Дополнительные условности:

1. Имя класса реализации - *SetImpl*
2. Класс должен иметь конструктор без параметров
3. Pull Request должен содержать в части тестирования проходящие тесты корректности на основе lincheck
4. В реализации не предполагается увидеть стандартные контейнеры из java.util.concurrent
5. Гарантировать исполнение на JDK 11
6. Изменение структуры данных должно происходить в разных частях независимо (то есть не должно быть contention гарантированно на одном элементе структуры данных, например - корне)
7. Число “холостых” аллокаций для потоков, у которых не проходит CAS должно быть минимизировано