### AmigoSet (1)

Давай напишем какую-нибудь коллекцию. Пусть это будет твой собственный Set.  
Пусть этот класс позволяет вставку **NULL**.

1. Создай класс **AmigoSet**. Пусть этот класс наследуется от **AbstractSet**.  
Этот сэт должен поддерживать интерфейсы **Serializable** и **Cloneable** (как же без этого??).  
Также очевидно, что он должен реализовывать интерфейс **Set**.

2. Этот класс должен работать с любыми типами, поэтому сделай его дженериком: добавь тип, например, **E**.  
Стандартные буквы, которые используют для дженериков - это **E** (**element**), **T** (**type**), **K**(**key**), **V** (**value**).  
Названия не принципиальны, но облегчают чтение кода.

3. Воспользуйся горячими клавишами Идеи и реализуй необходимые методы, оставь реализацию по умолчанию.

**AmigoSet (2)**

Изобретать механизм работы с хешем не будем, он уже реализован во многих коллекциях.

Мы возьмем коллекцию HashMap и воспользуемся ей.

1. Создай приватную константу Object PRESENT, которую инициализируй объектом Object, это будет наша заглушка.

2. Создай private transient поле HashMap<E,Object> map. Список ключей будет нашим сэтом, а вместо значений будем пихать в мапу заглушку PRESENT.

Напомню, нам нужны только ключи, а вместо значений для всех ключей будем вставлять PRESENT. Там же должно что-то быть :)

Посмотрим, что из этого получится :)

Коллекции обычно имеют несколько конструкторов, поэтому:

3. Создай конструктор без параметров, в котором инициализируй поле map.

4. Создай конструктор с одним параметром Collection<? extends E> collection.

Для инициализации поля map воспользуйся конструктором, в который передается Capacity.

Вычисли свою Capacity по такой формуле: максимальное из 16 и округленного в большую сторону значения (collection.size()/.75f)

Добавь все элементы из collection в нашу коллекцию.

Нужный метод добавления всех элементов у нас есть благодаря тому, что AbstractSet наследуется от AbstractCollection.

5. Напиши свою реализацию для метода метод add(E e): добавь в map элемент 'e' в качестве ключа и PRESENT в качестве значения.

Верни true, если был добавлен новый элемент, иначе верни false.

**Требования:**  
1. В классе AmigoSet должно быть создано и инициализировано private static final поле PRESENT типа Object.  
2. В классе AmigoSet должно быть создано private transient поле map типа HashMap.  
3. В классе AmigoSet должен быть реализован в соответствии с условием конструктор без параметров.  
4. В классе AmigoSet должен быть реализован в соответствии с условием конструктор с одним параметром типа Collection.  
5. Метод add должен добавлять новый элемент в map используя полученный параметр в качестве ключа и объект PRESENT в качестве значения.  
6. Метод add должен возвращать true в случае, если новый элемент был успешно добавлен, иначе – false

### AmigoSet (3)

Напиши свою реализацию следующих методов при условии, что нужно работать с ключами мапы:  
\* Iterator<E> iterator() - очевидно, что это итератор ключей. Получи множество ключей в map, верни его итератор  
\* int size() - это количество ключей в map, равно количеству элементов в map  
\* boolean isEmpty()  
\* boolean contains(Object o)  
\* void clear()  
\* boolean remove(Object o)

Ничего своего писать не нужно, используй то, что уже реализовано для множества ключей **map**.  
Используй Alt+Insert => Override methods

**Требования:**  
1. Метод iterator должен возвращать итератор для множества ключей поля map.  
2. Метод size должен возвращать то же, что и метод size поля map.  
3. Метод isEmpty должен возвращать true, если map не содержит ни одного элемента, иначе - false.  
4. Метод contains должен возвращать true, если map содержит анализируемый элемент, иначе - false.  
5. Метод clear должен вызывать метод clear объекта map.  
6. Метод remove должен удалять из map полученный в качестве параметра элемент.

### AmigoSet (4)

Твое собственное множество **AmigoSet** реализует интерфейс **Cloneable**. Однако, не клонируется правильно.  
Напиши свою реализацию метода Object clone(), сделай поверхностное клонирование.

\* Клонируй множество, клонируй **map**.  
\* В случае возникновения исключений выбрось **InternalError**.  
\* Убери лишнее пробрасывание исключения.

Расширь модификатор доступа до **public**.

**Требования:**  
1. В классе AmigoSet метод clone должен иметь уровень доступа public.  
2. В случае возникновения исключений в процессе клонирования должно быть брошено исключение InternalError.  
3. В классе AmigoSet метод clone должен быть реализован в соответствии с условием задачи.

### AmigoSet (5)

Твое собственное множество **AmigoSet** реализует интерфейс **Serializable**. Однако, не сериализуется правильно.

1. Реализуй свою логику сериализации и десериализации.  
Вспоминай, какие именно приватные методы нужно добавить, чтоб сериализация пошла по твоему сценарию.  
**Для сериализации:**  
\* сериализуй сет  
\* сериализуй **capacity** и **loadFactor** у объекта map, они понадобятся для десериализации.  
Т.к. эти данные ограничены пакетом, то воспользуйся утилитным классом **HashMapReflectionHelper**, чтобы достать их.

**Для десериализации:**  
\* вычитай все данные  
\* создай мапу используя конструктор с **capacity** и **loadFactor**

2. Помнишь, что такое **transient**?

**Требования:**  
1. В классе AmigoSet должен содержаться private метод writeObject с одним параметром типа ObjectOutputStream.  
2. В классе AmigoSet должен содержаться private метод readObject с одним параметром типа ObjectInputStream.  
3. Объект сериализованный с помощью метода writeObject должен быть равен объекту десериализованному с помощью метода readObject.  
4. В методе writeObject должен быть вызван метод defaultWriteObject на объекте типа ObjectOutputStream полученном в качестве параметра.  
5. В методе readObject должен быть вызван метод defaultReadObject на объекте типа ObjectInputStream полученном в качестве параметра.