Задание 1. Классы коллекций

Изучите классы реализации коллекций и заполните следующую таблицу

	Ordering (в смысле порядка добавления)	**Random Access**	**Key-Value Pairs**	**Allows Duplicates**	**Allows Null Values**	**Thread Safe**	**Blocking Operations**
ArrayList	YES	YES	NO	YES	YES	NO	NO
LinkedList	YES	YES	NO	YES	YES	NO	NO
ArrayDeque	YES	NO	NO	YES	NO	NO	NO
HashSet	NO	NO	NO	NO	YES	NO	NO
LinkedHashSet	YES	NO	NO	NO	YES	NO	NO
TreeSet	NO	NO	NO	NO	YES	NO	NO
HashMap	NO	YES	YES	NO(keys)	YES	NO	NO
LinkedHashMap	YES	YES	YES	NO(keys)	YES	NO	NO
TreeMap	NO	YES	YES	NO(keys)	NO (только в пустой единожды)	NO	NO
CopyOnWriteArrayList	YES	YES	NO	YES	YES	YES	YES
CopyOnWriteArraySet	NO	NO	NO	NO	YES	YES	YES
ConcurrentHashSet	NO	NO	NO	NO	YES	YES	YES
ConcurrentHashMap	NO	YES	YES	NO	YES	YES	YES
BlockingQueue	YES	NO	NO	YES	NO	YES	YES

Задание 2. Использование Map

Создать “универсальный” класс, позволяющий получить значение из любого properties-файла. Физическое чтение файла должно происходить только один раз. Результаты чтения храните в коллекции типа Map. Ответьте на вопрос: как ведет себя map-коллекция если в нее добавить элемент с ключом, который уже присутствует?

/реализация в ДЗ 5/

Ответ: If you try to add a duplicate key to a HashMap it's new value will replace the key's previous value.

Задание 3. Ссылки на коллекции

Определена иерархия классов:

```
class MedicalStaff{}
class Doctor extends MedicalStaff{}
class Nurse extends MedicalStaff{}
class HeadDoctor extends Doctor{}
```

Укажите корректные и некорректные операторы. Дайте ответу пояснение.

	CORRECT	**INCORRECT**
Doctor doctor1 = new Doctor();	+	
Doctor doctor2 = new MedicalStaff();		- This assignment isn't allowed. Impossible to create an instance of a subclass from a super class's constructor
Doctor doctor3 = new HeadDoctor();	+ It's ok to declare an instance of HeadDoctor as a Doctor object, because HeadDoctor extends Doctor	
Object object1 = new HeadDoctor();	+ Because HeadDoctor extends Object	
HeadDoctor doctor5 = new Object();		- Impossible to create an instance of a subclass from a super class's constructork
Doctor doctor6 = new Nurse();		- Neither Doctor nor Nurse extend each other
Nurse nurse = new Doctor();		- Neither Doctor nor Nurse extend each other
Object object2 = new Nurse();	+ Because Nurse extends Object	
List<Doctor> list1= new ArrayList<Doctor>();	+ generic types are the same	
List<MedicalStaff> list2 = new ArrayList<Doctor>();		- This assignment isn't allowed. If we declare a reference variable List<MedicalStaff> list2, whatever we assign it to must be of generic type MedicalStaff
List<Doctor> list3 = new ArrayList<MedicalStaff>();		- This assignment isn't allowed. If we declare a reference variable List<Doctor> list3, whatever we assign it to must be of generic type Doctor
List<Object> list4 = new ArrayList<Doctor>();		-

		This assignment isn't allowed. If we declare a reference variable List<Object> list4, whatever we assign it to must be of generic type Object
List<Object> list5 = new ArrayList<Object>();	+ generic types are the same	

Задание 4. Применение коллекций

Заполните таблицу

	Основная функциональность	**Примеры типичного использования**
Set	The Set interface models the mathematical Set abstraction. It's a collection of objects that doesn't contain duplicate element and the elements are returned in no particular order. To determine the equality of objects, Set uses its method equals(). For two elements, say e1 and e2, if e1.equals(e2) returns true, Set doesn't add both these elements. Set defines methods to add(add(E e), addAll(Collection<? extends E> c)) and remove its elements(:remove(Object o),removeAll(Collection<? > c)). It also defines methods to queryitself for the occurrence of specific objects. Because it indirectly implements theIterable interface, it includes method iterator() to retrieve an Iterator. It alsoincludes methods to convert it into an array (toArray())	**HashSet** class - is implemented using a HashMap, it uses hashing method to store and retrieve storing, accessing an object's hashCode value to determine the bucket in which it should be stored. A **LinkedHashSet** offers the benefits of a HashSet combined with a LinkedList. It maintains a double-linked list running through its entries.As with a LinkedList, you can retrieve objects from a LinkedHashSet in the order of their insertion. A **TreeSet** stores all its unique elements in a sorted order. The elements are ordered either on their natural order (achieved by implementing the Comparable interface) or by passing a Comparator, while instantiating a TreeSet. If you fail to specify either of these, TreeSet will throw a runtime exception when you try to add an object to it.TreeSet uses a *Black-Red binary tree*. This tree modifies itself as you add more values to it so it has the least number of levels and the values are distributed as evenly as possible.
List	An ordered collection of objects. It allows to store duplicate elements. In a List you can control the position where you want to store an elements, that is why it defines overloaded methods to add, remove and retrive elements at a particular position. <u>Modification methods:</u> -Add elements: add(E e), addAll(Collection<? extends E> c)	**ArrayList** class - resizable array to store elements(insertion,deletion O(n), random access O(1)) **LinkedList** class - double-linked list (insertion,deletion O(1), random access O(n)) - a good choice if implementation of stack or queue is needed.

	-Remove elements:remove(Object o),removeAll(Collection<? > c) -Modify: set(int index, E c) <u>Query methods</u>: subList(int from, int to) lastIndexIf(Object o) <u>Miscellaneous:</u> toArray() toArray(T[] a) <u>Iterator methods:</u> iterator() listIterator() listIterator(int index) (…)	
Queue	METHODS: add elements in tail: (add(E e), offer(E e) query and remove from head: poll() /if isEmpty() returns null/, remove()/if isEmpty() returns NoSuchElementException/ query elements: peek() /same/, element() /same/	The Deque interface extends the Queue interface. As a double-ended queue, a Deque can work as both a *queue* and a *stack*.
Map	Doesn't extend the Collection interface.A Map doesn't allow the addition of duplicate keys. Items added to a Map aren't ordered. The retrieval order of items from a Map object isn't guaranteed to be the same as its insertion order. The Map interface declares methods to add or delete a key-value pair or query the existence of a key or value. It also defines methods to retrieve the set of keys, values, and key-value pairs. METHODS: void clear() boolean containsKey(Object k), containsValue(Object v) Set<Map.Entry<K,V>>entrySet() V get(Object k) V put(K k, V v) Set<K> keySet() Collection<V> values() (...)	Map is used to add or retrieve key-value pairs in O(1) (because of usage of hashing functions).