4. Обобщения и лямбды

Программирование на Java

Автор курса: Федор Лаврентьев

Лектор: Виктор Яковлев

МФТИ, 2016-2017

Массивы

Массив

```
int[] ints = new int[10];
assert ints.length == 10;
for (int i = 0; i < ints.length; ++i) {
  ints[i] = i * i;
}

Object arr = ints; // Array is an object

final Object[] objects = new Object[1];
objects = new Object[1]; // WRONG
objects[0] = arr;        // But right o_O
```

Инициализация массивов

```
double[] coeffs = {3.0, 4.0, 5.0};

int[][] intMatrix = new int[3][4];
// 0 -> [0, 0, 0, 0]
// 1 -> [0, 0, 0, 0]
// 2 -> [0, 0, 0, 0]

char[][] words = {{‘a’, ‘b’, ’c’}, {‘d’, ‘e’}};
// 0 -> [‘a’, ‘b’, ‘c’]
// 1 -> [‘d’, ‘e’]
```

Обобщённые типы

Подробнее здесь:
https://docs.oracle.com/javase/tutorial/java/generics/index.html

Логика не зависит от типов аргументов

```
public class Pair {
  final Object left;
  final Object right;

  public Pair(Object left, Object right) {
    this.left = left;
    this.right = right;
  }

  public Object getLeft() { return left; }
  public Object getRight() { return right; }
}
```

Уточнение типов усложняет код

```
Pair pair = new Pair(42, “Forty two”);

Object leftObj = pair.getLeft();
Integer left = (Integer) leftObj;

String right = (String) pair.getRight();
```

Приведение типов потенциально опасно

```
public long extractLongLeft(Pair pair) {
  Object leftObj = pair.getLeft;
  if (!(leftObj instanceof Long)) {
    throw new IllegalArgumentException();
  }
  Long leftLong = (Long) leftObj;
  return leftLong.longValue(); // Unnecessary
}
```

Generics (обобщения)

```
public class Pair<A, B> {
  private final A left;
  private final B right;

  public Pair(A left, B right) {
    this.left = left;
    this.right = right;
  }

  public A getLeft() { return left; }
  public B getRight() { return right; }
}
```

Generics (обобщения)

```
Pair<Integer, String> p1 =
    new Pair<Integer, String> (42, “Forty two”);

String right1 = p1.getRight(); // “Forty two”

Pair<Long, String> p2 = new Pair<>(0L, “Null”);

Long left1 = p2.getLeft(); // 0L
```

Подтипы

```
public interface Comparable<T> {
  boolean geq(T other) default {
    this.hashCode() >= other.hashCode();
  }
}

public class Comparables {
  public static <T extends Comparable> T min(T a, T b) {
    if (a.geq(b)) return b else return a;
  }

  public static <T extends Comparable> T max(T a, T b) {
    if (a.geq(b)) return a else return b;
  }
}
```

Подтипы

```
import static Comparables.*;

class SortedPair<T extends Comparable>
    extends Pair<T, T> {

  public SortedPair(T a, T b) {
    super(min(a, b), max(a, b))
  }
}
```

Лямбды

Функциональный интерфейс

```
public interface Runnable {
  void run();
}

public class HelloRunnable implements Runnable {
  @Override
  public void run() {
    System.out.println(“Hello!”);
  }
}

Thread t = new Thread(new HelloRunnable());
t.start();
```

Функциональный интерфейс

```
public interface Runnable {
  void run();
}

Thread t = new Thread(new Runnable() {
  @Override
  public void run() {
    System.out.println(“Hello!”);
  }
});


t.start();
```

Функциональный интерфейс

```
public interface Runnable {
  void run();
}

Thread t = new Thread(() -> {
  System.out.println(“Hello!”);
});
```

```
t.start();
```

Функциональный интерфейс

```
@FunctionalInterface
public interface Runnable {
  void run();
}

Thread t = new Thread(() -> {
  System.out.println(“Hello!”);
});






t.start();
```

Лямбда-выражение

```
@FunctionalInterface
interface Comparator<T> {
  int compare (T first, T second);
}


Comparator<String> c1 =
    (String a, String b) -> a.length() – b.length();


Comparator<String> c2 = (a, b) -> a.length() – b.length();


Comparator<String> c3 = (a, b) -> {
  int lengthA = (a == null) ? 0 : a.length();
  int lengthB = (b == null) ? 0 : b.length();
  return lengthA – lengthB;
}
```

Основные структуры данных

Упорядоченный список (List)

```
List<Dog> dogs = Arrays.asList(
  new Dog(“Billy”, “Bloodhound”),
  new Dog(“Jackie”, “Bulldog”)
);

List<Animal> animals = new ArrayList<>(4);
animals.addAll(dogs);
animals.add(0, new Dog(“Pooker”, “Mongrel”));
animals.add(new Cat(“Snowee”, “Birman”));

List<Cat> cats = new LinkedList<>();
cats.add(new Cat(“Snowee”, “Birman”));

animals.containsAll(cats); // true
animals.get(0); // Pooker
```

List processing

```
List<Animal> animals = ...;

int totalTeeth = 0
for (int i = 0; i <= animals.size(); ++i) {
  Animal a = animals.get(i);
  if (!(a instanceof Dog)) {
    continue;
  }
  Dog d = (Dog) a;
  totalTeeth += d.getTeethCount();
}
```

Stream

```
List<Animal> animals = ...;

Stream<Animal> stream = animals.stream();

int totalTeeth = stream
  .filter(a -> a instanceof Dog) // Stream<Animal>
  .map(a -> (Dog) a)             // Stream<Dog>
  .mapToInt(Dog::getTeethCount)  // IntStream
  .sum();                        // int
```

Уникальное множество (Set)

```
Set<Set> unique = new TreeSet(“Billy”, “Annett”, “Jane”, “Мишаня”,
“Pooker”, “Billy”);

unique.size(); // 5 (не 6!)

unique.headSet(“July”).foreach(System.out::println);
// Annett, Billy, Jane

unique.contains(“Annett”); // true
```

~~Отображение~~ Словарь (Map)

```
Map<String, Animal> byName = new HashMap<>();
animals.stream()
  .foreach(a -> byName.put(a.getName(), a))

byName.contains(“Bloody Joe”); // false

Animal pooker = byName.get(“Pooker”);

byName.entrySet().foreach(System.out::println);
// random order
```

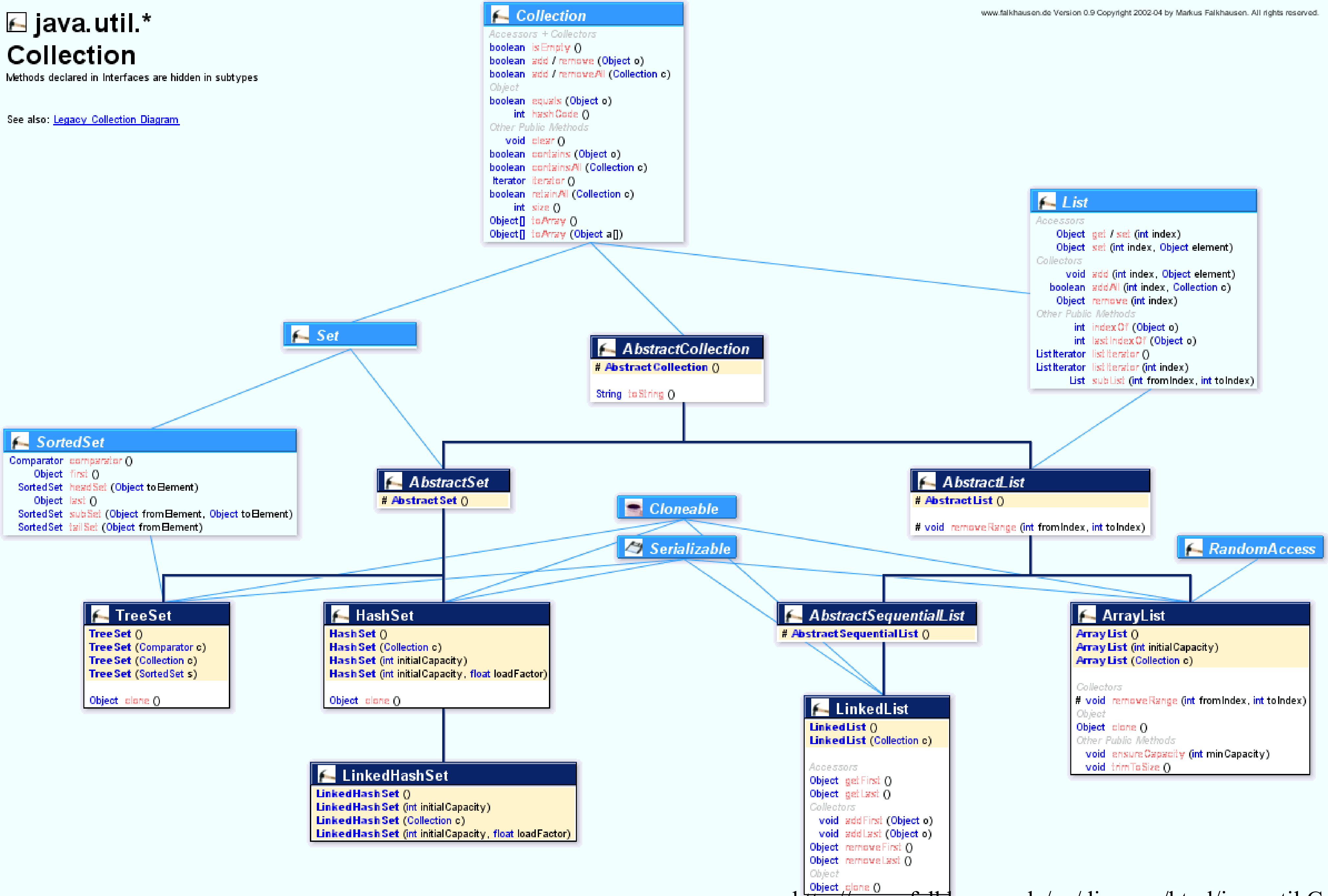
Iterable

```
interface Iterable<T> {
  Iterator<T> iterator();
  Spliterator<T> spliterator();
  void foreach(Consumer<? super T> consumer>);
}

List<Animal> animals = ...;

for (Iterator<Animal> it = animals.iterator();it.hasNext();) {
  Animal a = it.next();
  ...
}

for (Animal a: animals) { ... }

animals.foreach(a -> ...)
```

java.util.*
Collection

Methods declared in Interfaces are hidden in subtypes

See also: Legacy Collection Diagram



Collection
Accessors + Collectors
boolean isEmpty ()
boolean add / remove (Object o)
boolean add / removeAll (Collection c)
Object
boolean equals (Object o)
int hashCode ()
Other Public Methods
void clear ()
boolean contains (Object o)
boolean containsAll (Collection c)
Iterator iterator ()
boolean retainAll (Collection c)
int size ()
Object[] toArray ()
Object[] toArray (Object a[])

List
Accessors
Object get / set (int index)
Object set (int index, Object element)
Collectors
void add (int index, Object element)
boolean addAll (int index, Collection c)
Object remove (int index)
Other Public Methods
int indexOf (Object o)
int lastIndexOf (Object o)
ListIterator listIterator ()
ListIterator listIterator (int index)
List subList (int fromIndex, int toIndex)

Set

AbstractCollection
# AbstractCollection ()
String toString ()

SortedSet
Comparator comparator ()
Object first ()
SortedSet headSet (Object toElement)
Object last ()
SortedSet subSet (Object fromElement, Object toElement)
SortedSet tailSet (Object fromElement)

AbstractSet
# AbstractSet ()

Cloneable

Serializable

AbstractList
# AbstractList ()
# void removeRange (int fromIndex, int toIndex)

RandomAccess

TreeSet
TreeSet ()
TreeSet (Comparator c)
TreeSet (Collection c)
TreeSet (SortedSet s)
Object clone ()

HashSet
HashSet ()
HashSet (Collection c)
HashSet (int initialCapacity)
HashSet (int initialCapacity, float loadFactor)
Object clone ()

AbstractSequentialList
# AbstractSequentialList ()

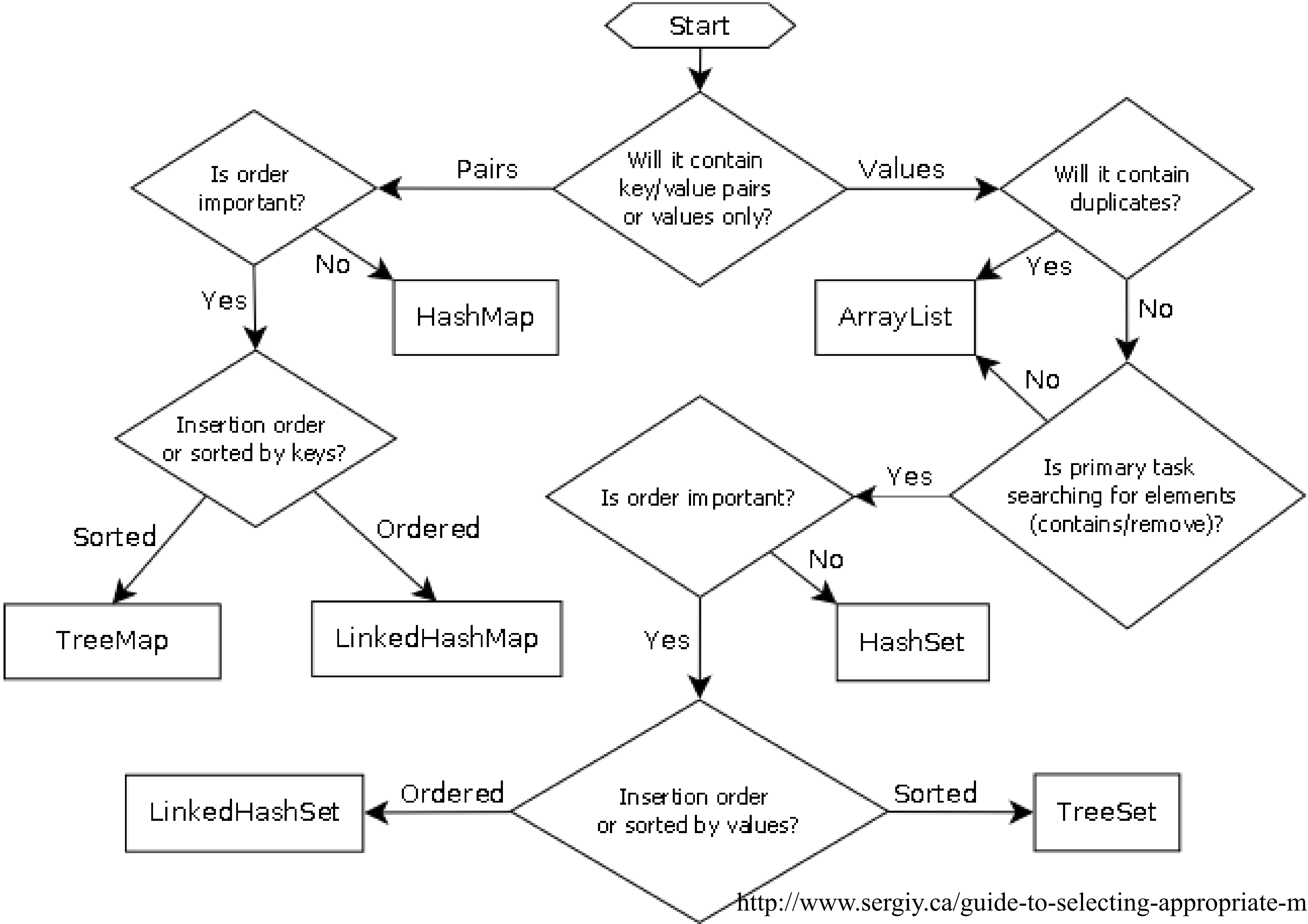
ArrayList
ArrayList ()
ArrayList (int initialCapacity)
ArrayList (Collection c)
Collectors
# void removeRange (int fromIndex, int toIndex)
Object
Object clone ()
Other Public Methods
void ensureCapacity (int minCapacity)
void trimToSize ()

LinkedHashSet
LinkedHashSet ()
LinkedHashSet (int initialCapacity)
LinkedHashSet (Collection c)
LinkedHashSet (int initialCapacity, float loadFactor)

LinkedList
LinkedList ()
LinkedList (Collection c)
Accessors
Object getFirst ()
Object getLast ()
Collectors
void addFirst (Object o)
void addLast (Object o)
Object removeFirst ()
Object removeLast ()
Object
Object clone ()

http://www.falkhausen.de/en/diagram/html/java.util.Collection.html

Java Map/Collection Cheat Sheet


Start
Will it contain key/value pairs or values only?
Pairs
Values
Is order important?
No
HashMap
Yes
Insertion order or sorted by keys?
Sorted
TreeMap
Ordered
LinkedHashMap
Will it contain duplicates?
Yes
ArrayList
No
Is primary task searching for elements (contains/remove)?
No
Yes
Is order important?
No
HashSet
Yes
Insertion order or sorted by values?
Ordered
LinkedHashSet
Sorted
TreeSet

http://www.sergiy.ca/guide-to-selecting-appropriate-map-collection-in-java/