Advanced Programming

Lecture 6: Java Collections Framework++

Tommi Tervonen

Econometric Institute, Erasmus School of Economics

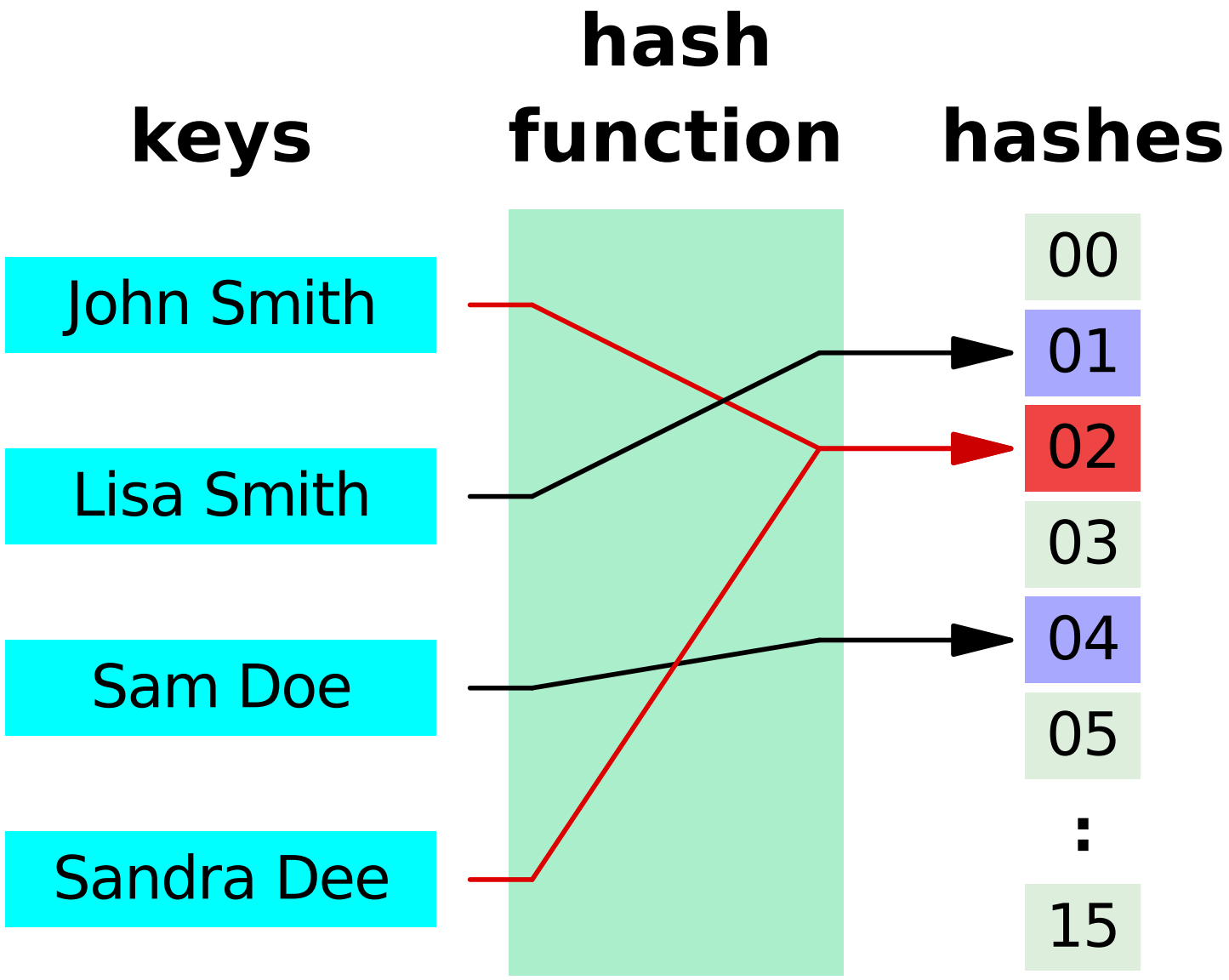
Object identity

- Object identity is defined with `equals`, default implementation: reference equality

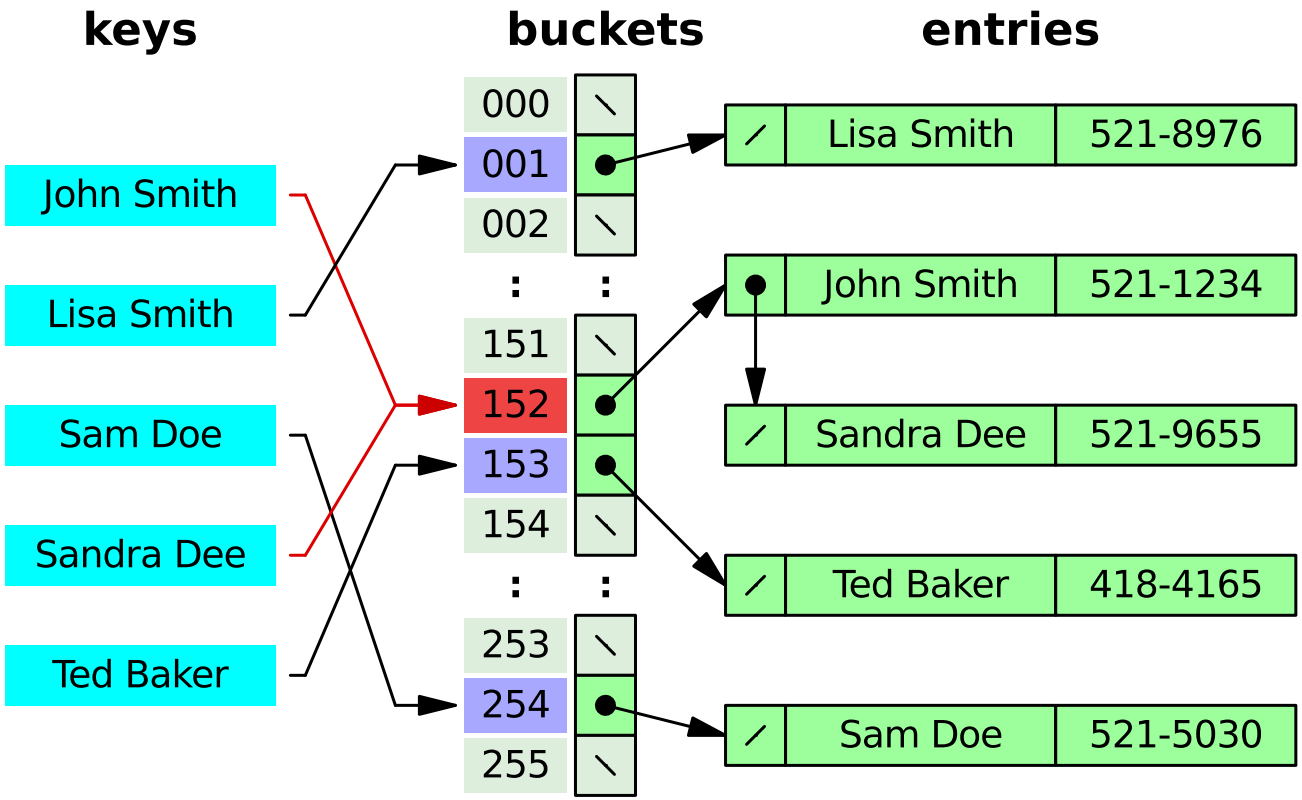
```
public class Student {
  ...
  public boolean equals(Object o) {
    if (o instanceof Student) {
      Student so = (Student) o;
      return so.getName().equals(getName());
    }
    return false;
  }
}

Student s1 = new Student("tommi", 121212);
Student s2 = new Student("tommi", 343434);
Assert.assertEquals(s1, s2); // passes
```

How to store unique objects efficiently?

Hashing

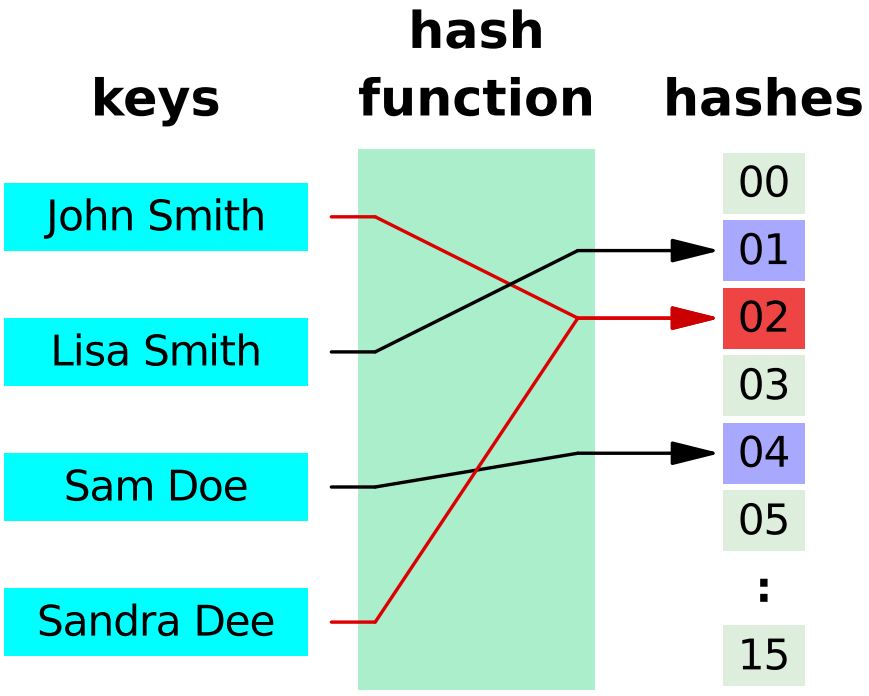


Hashtable buckets with linked lists



- `bucketIndex = obj.hashCode() % nrBuckets`
- Search within bucket uses `equals`

hashCode



- Good `hashCode` implementation spreads the objects evenly across the `int` range
- `29 * name.hashCode() + number`
- Use primes as multiplication factors

- Objects must be invariant re: the hashed fields
- `a.equals(b)` $\Longrightarrow$ `a.hashCode()==b.hashCode()`

Generic classes

- Explicit downcasting is dangerous and should be avoided
- Generic classes allow to specify explicitly a type that varies
- The actual type information is added during compilation time (example: compile-time vs run-time type checking)

```
public class Sorter<T extends Keyable> {
  public void sort(T[] arr) { ... }
}
```

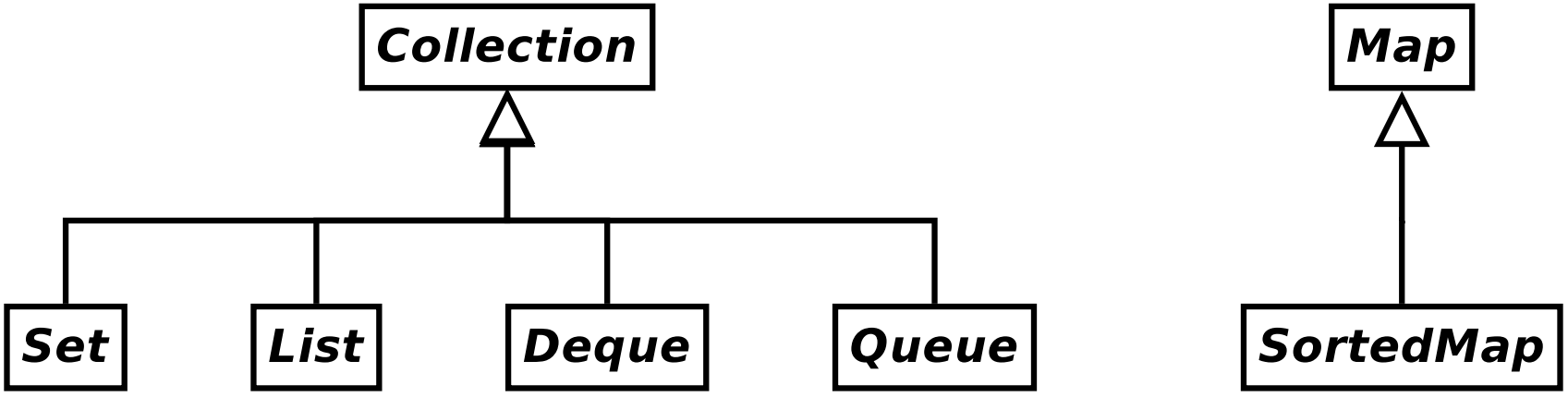
- Auto-boxing of primitive types to the corresponding object ones

```
Double d = 2.0;
double pd = new Double(3.0);
```

Java Collections Framework

- Interfaces (abstract data types)
- Concrete classes (and some abstract ones)
- Iterators
- Algorithms

JCF interfaces



Collection extends Iterable

- Unordered bag of items
- `add`, `remove`, `isEmpty`, `size`, `contains`, `clear`

```
Collection<String> c = new ... // alloc
... // fill
String[] arr = c.toArray(new String[0]); // !
```

List extends Collection

- Ordered bag of items
- Random access
- `add(T, int)`, `get(int)`, `remove(int)`

Set extends Collection

- Unordered set of items
- `SortedSet` for an ordered set of items (still no random access)

Queue and Deque

- Queue, Double Ended Queue
- Queue not necessarily implements FIFO ordering
- `peek`, `poll`, `remove`

Map

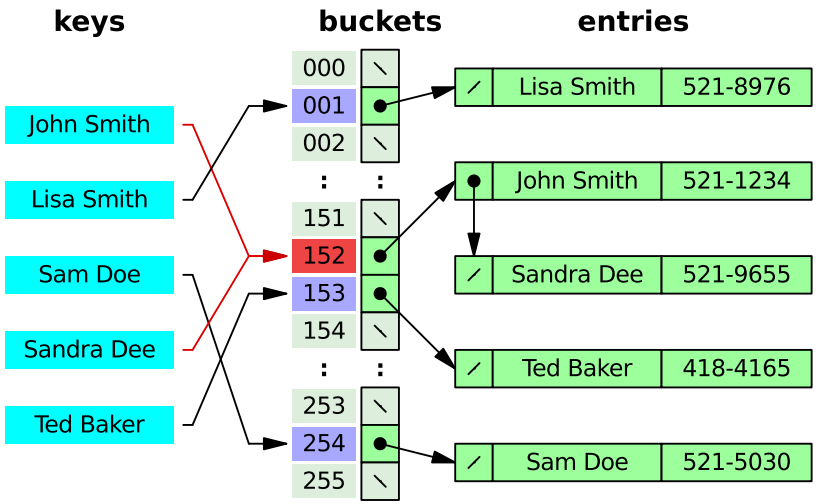
- (key, value) pairs
- aka associative array and dictionary

List implementations

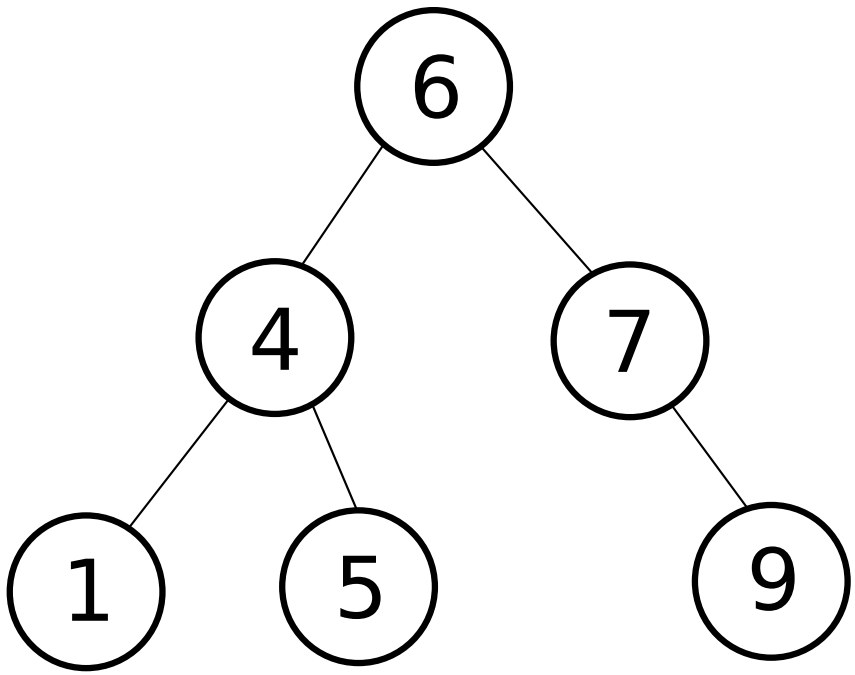
- ArrayList
- LinkedList

Set implementations

HashSet



TreeSet



Comparable interface

```
public class Student implements Comparable<Keyable>
 {
   ...
   public int compareTo(Keyable k) {
     if (getKey() > k.getKey()) {
       return 1;
     } else if (getKey < k.getKey()) {
       return -1;
     }
     return 0;
   }
}
```

`Map` implementations

- `TreeMap` (red-black tree, used for TreeSet)
- `HashMap`

Algorithms

- `Collections.sort`
- `Collections.shuffle`
- `Collections.reverse .fill .copy .swap .addAll`
- `Collections.binarySearch`
- `Collections.frequency .disjoint`
- `Collections.min .max`
- With `List` or `Collection`

Design patterns

- More often than not we encounter similar problems in building programs
- General solutions to such often encountered problems are called **design patterns**

Observer pattern