Midterm Info

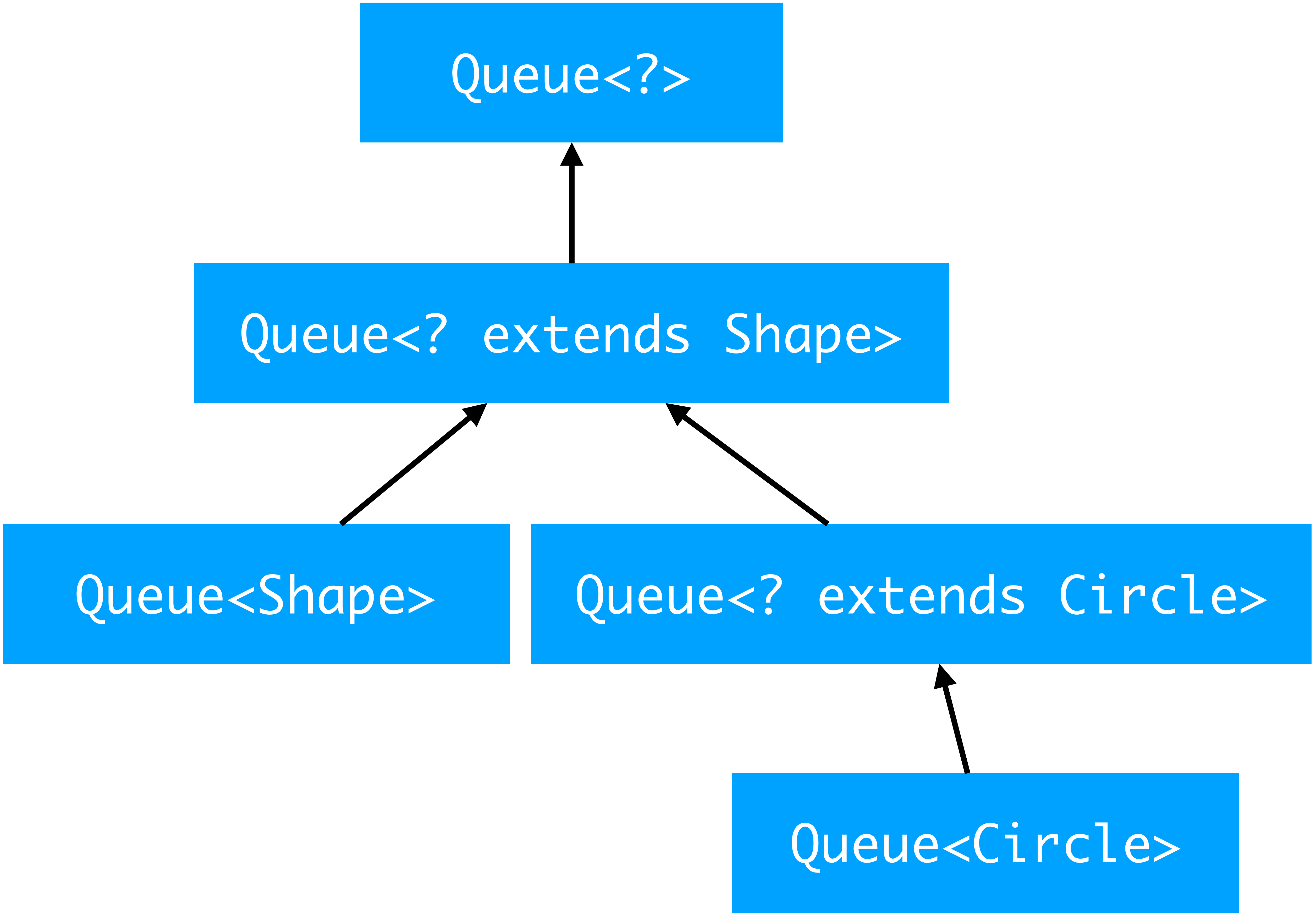
- **Date**: 5 March 2018
  Monday after recess week
- **Time**: 1000am - 1130am 90 minutes
- **Venue**: MPSH M2C (Multipurpose Hall 2, Section C)

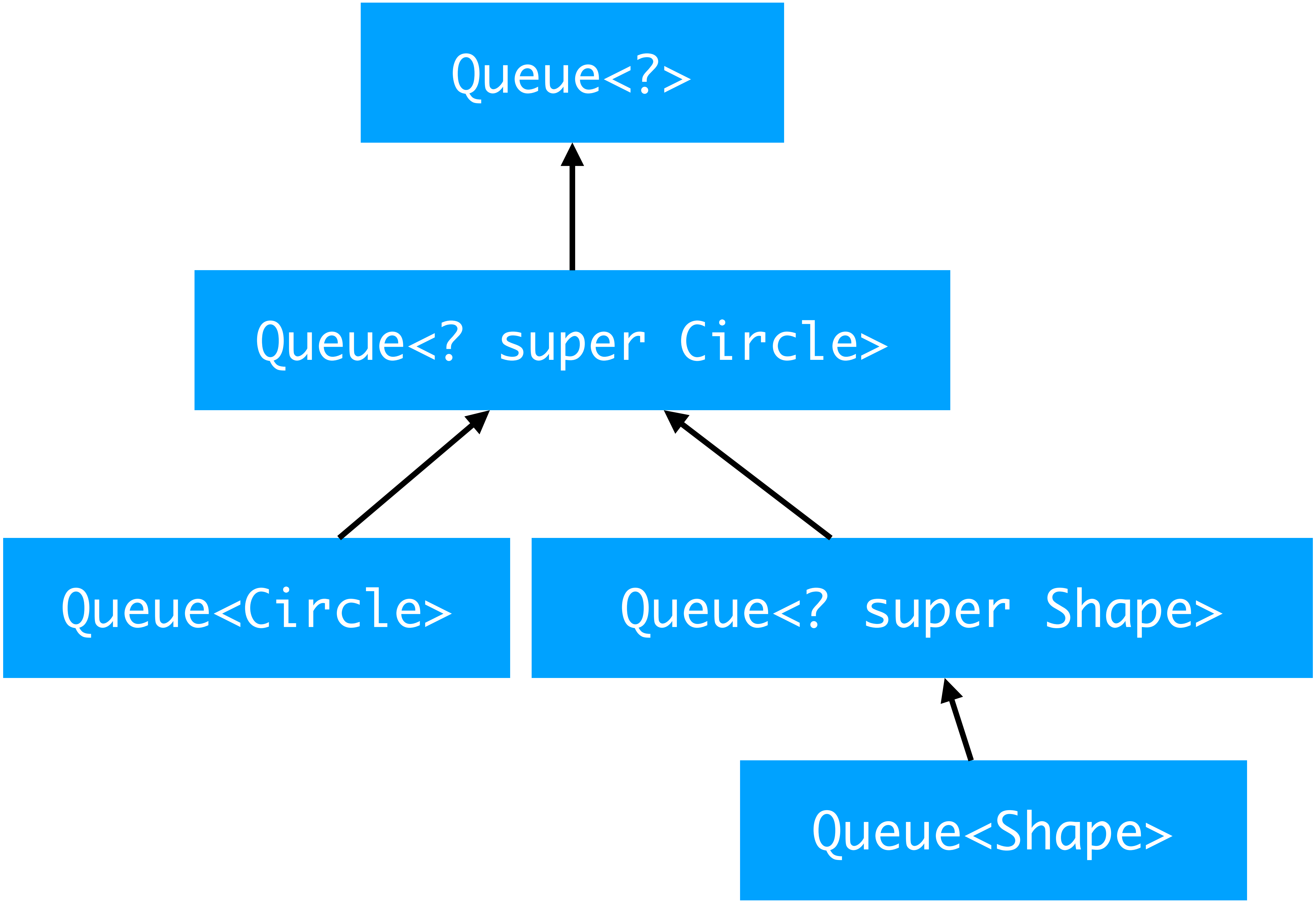
Previously, in cs2030..

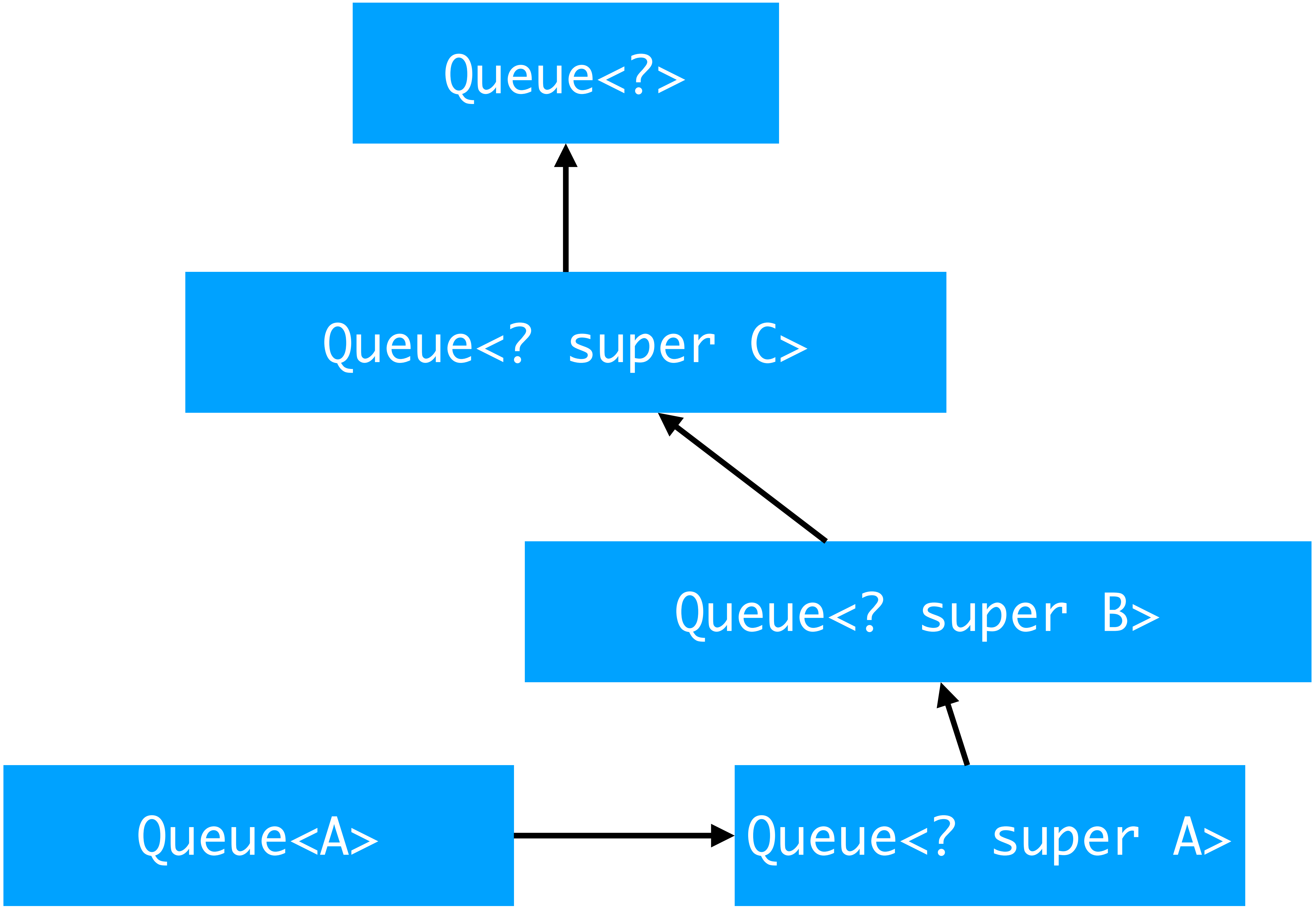
```
class Queue<T> {
    :
}
```

```
Queue<Point> q = new Queue<>(4);
```

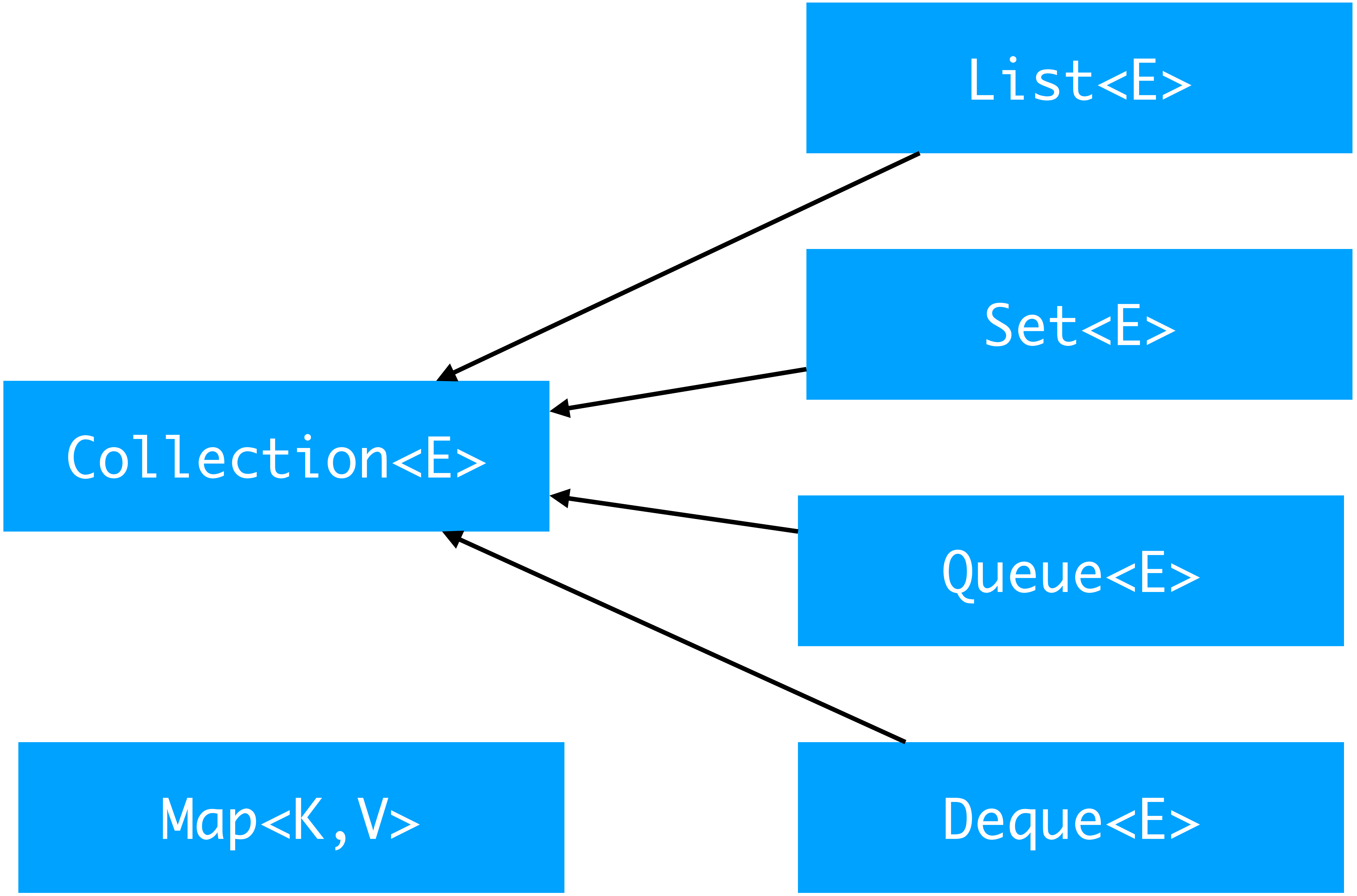
`Queue<T>` is the *generic class*
`Queue<Point>` is a *parameterised type*
`T` is the *type parameter*
`Point` is the *type argument*


Queue<?>
Queue<? extends Shape>
Queue<Shape>
Queue<? extends Circle>
Queue<Circle>

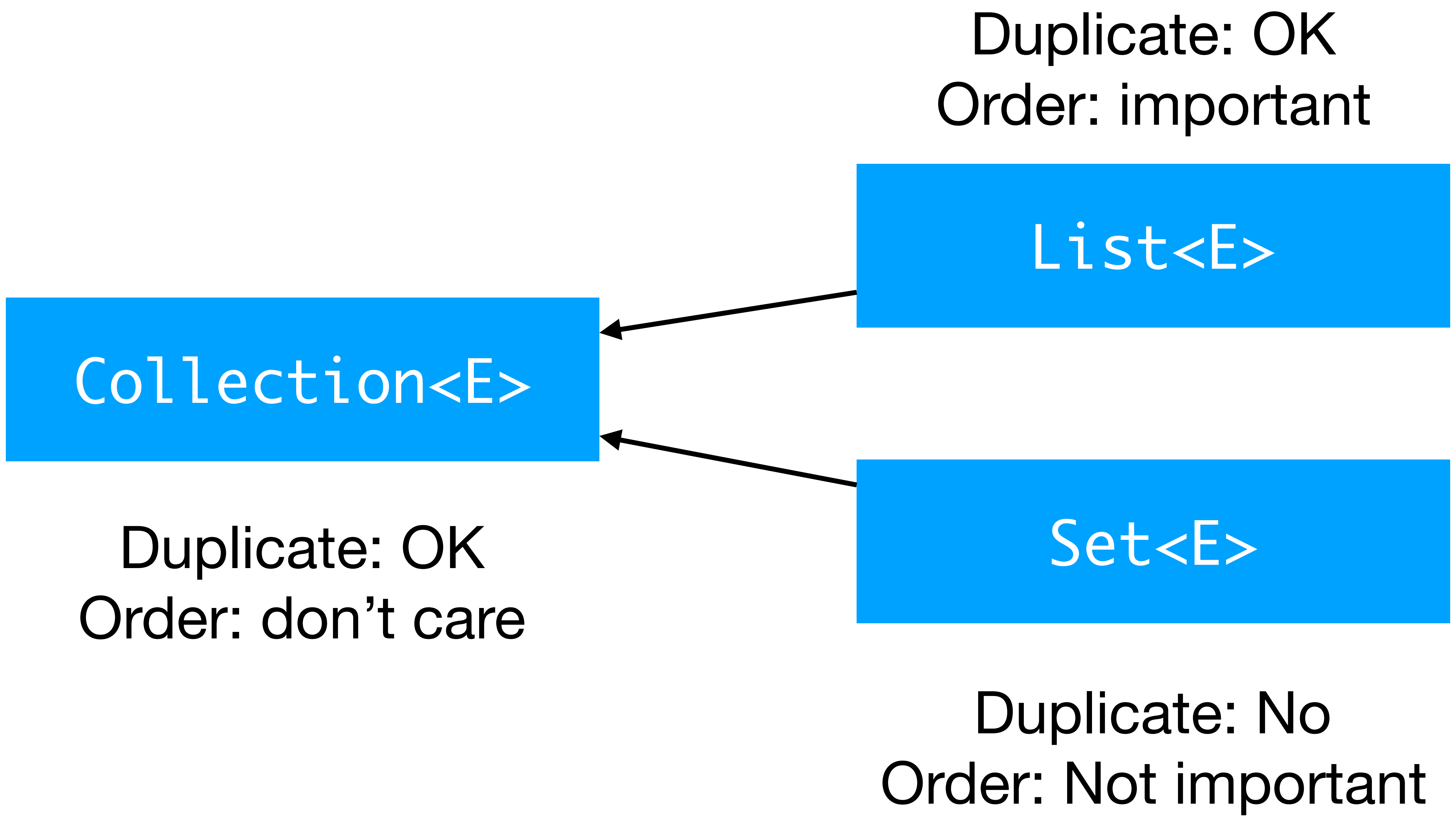

Queue<?>
Queue<? super Circle>
Queue<Circle>
Queue<? super Shape>
Queue<Shape>


Queue<?>
Queue<? super C>
Queue<? super B>
Queue<A>
Queue<? super A>

Java Collections


List<E>
Set<E>
Collection<E>
Queue<E>
Map<K,V>
Deque<E>

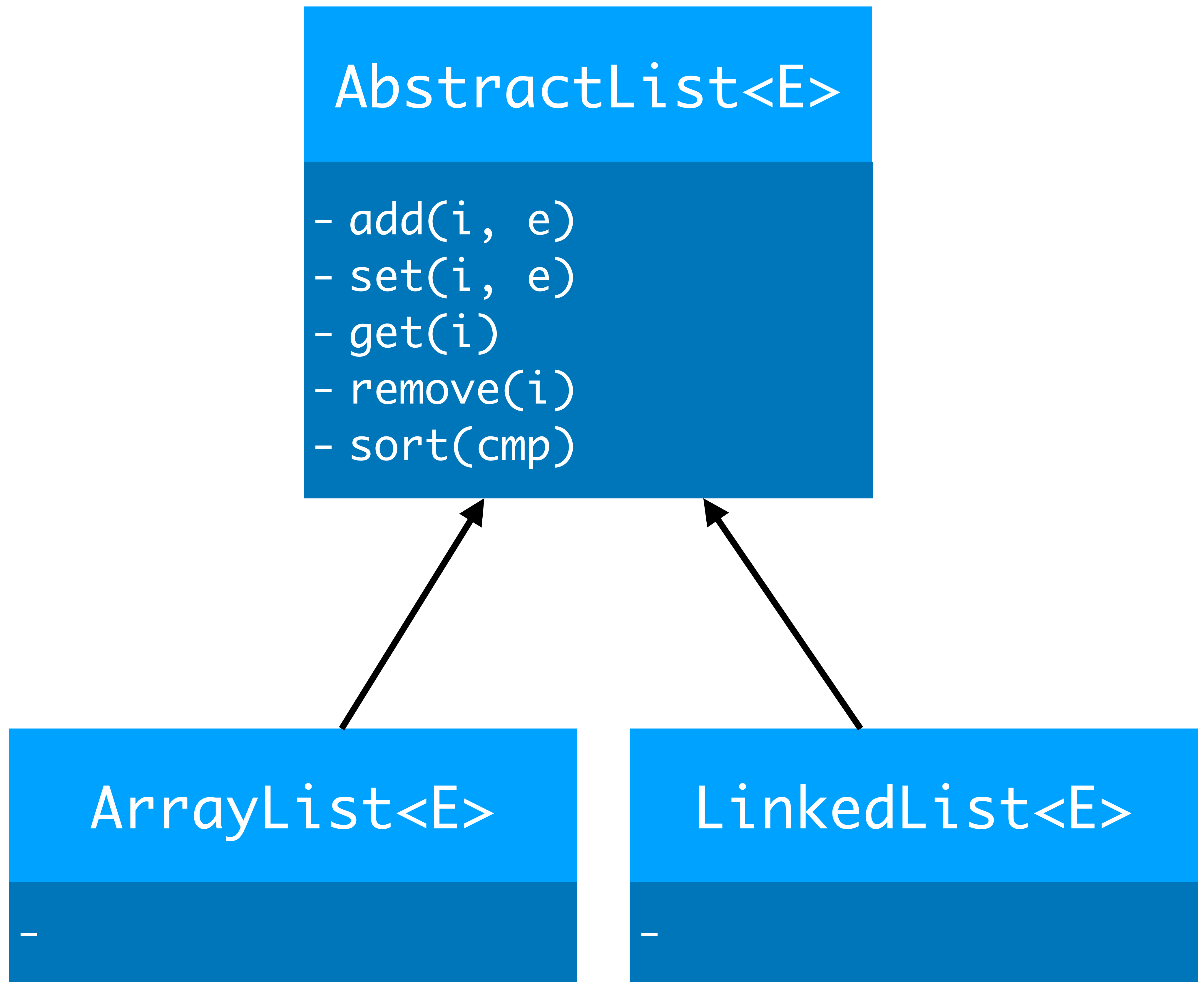
```java
public interface Collection<E> extends
    Iterable<E> {
  boolean add(E e);
  boolean contains(Object o);
  boolean remove(Object o);
  void clear();
  boolean isEmpty();
  int size();
  Object[] toArray();
  <T> T[] toArray(T[] a);
  boolean addAll(Collection<? extends E> c);
  boolean containsAll(Collection<?> c);
  boolean removeAll(Collection<?> c);
  boolean retainAll(Collection<?> c);
}
```

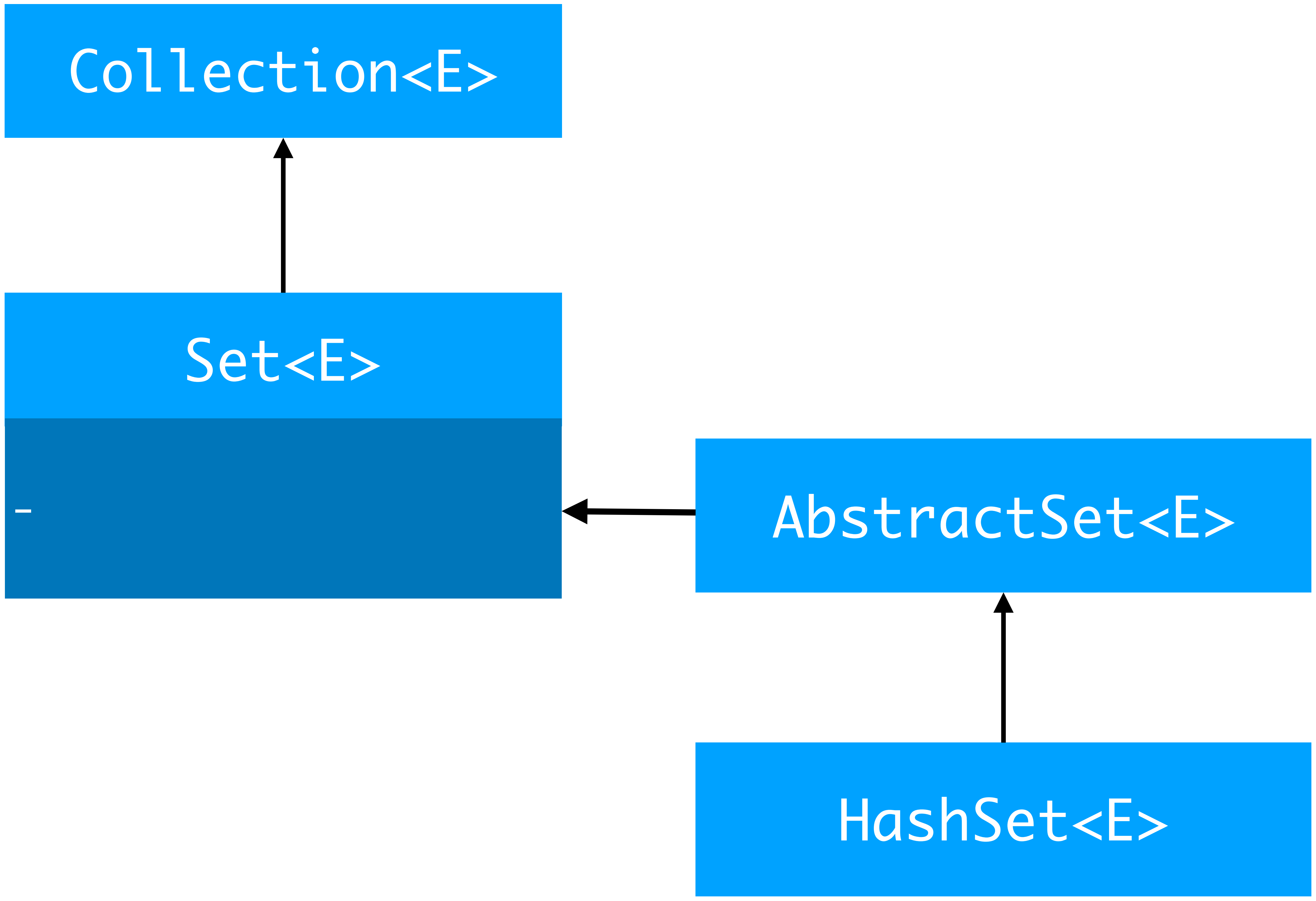

Duplicate: OK
Order: important
List<E>
Collection<E>
Set<E>
Duplicate: OK
Order: don’t care
Duplicate: No
Order: Not important

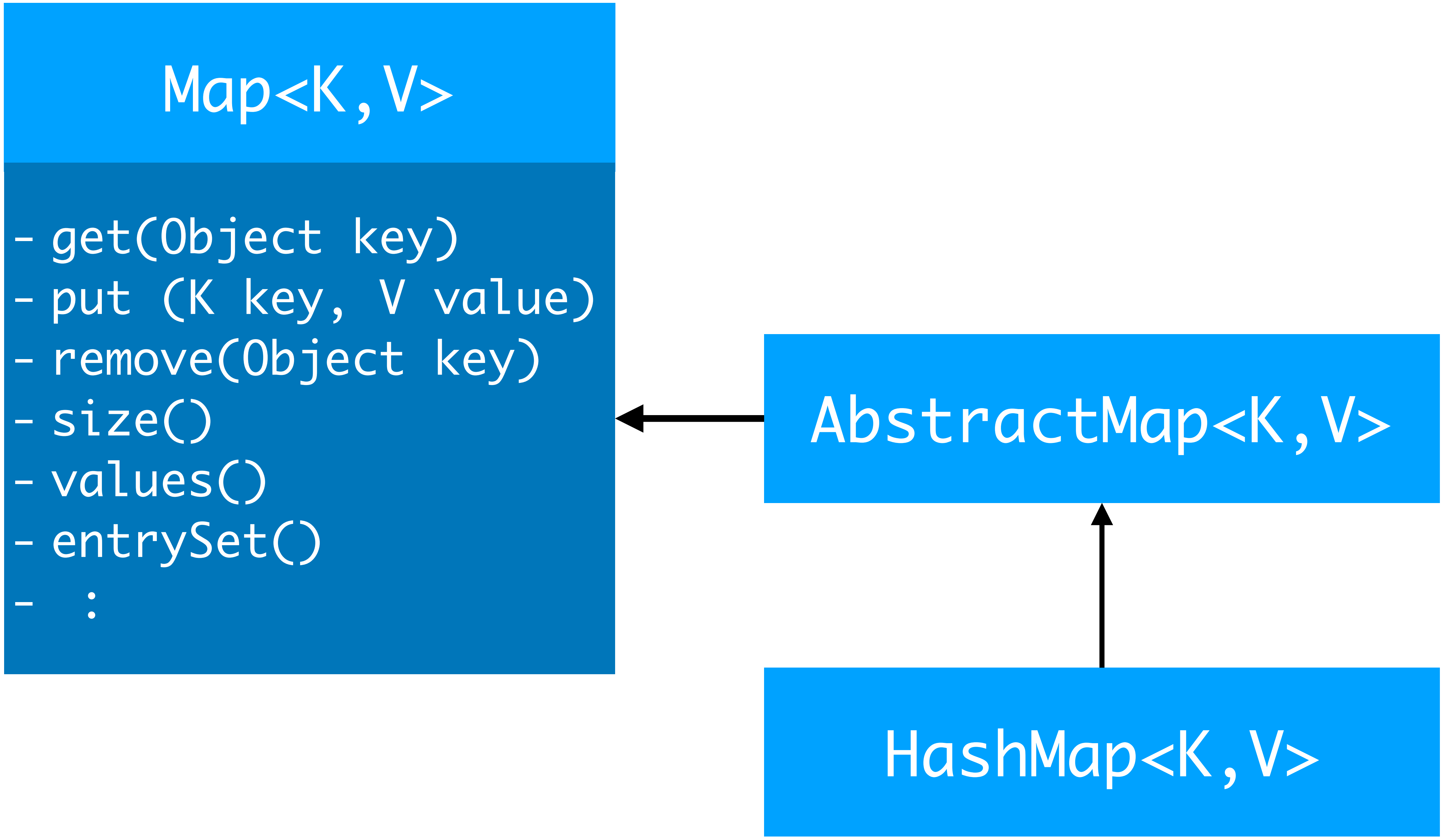
```
public interface Iterable<T> {
  Iterator<T> iterator();
}

public interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove();
}
```

```
public interface List<E> {
     :
   default void sort(Comparator<? super E> c)
     :
}
```

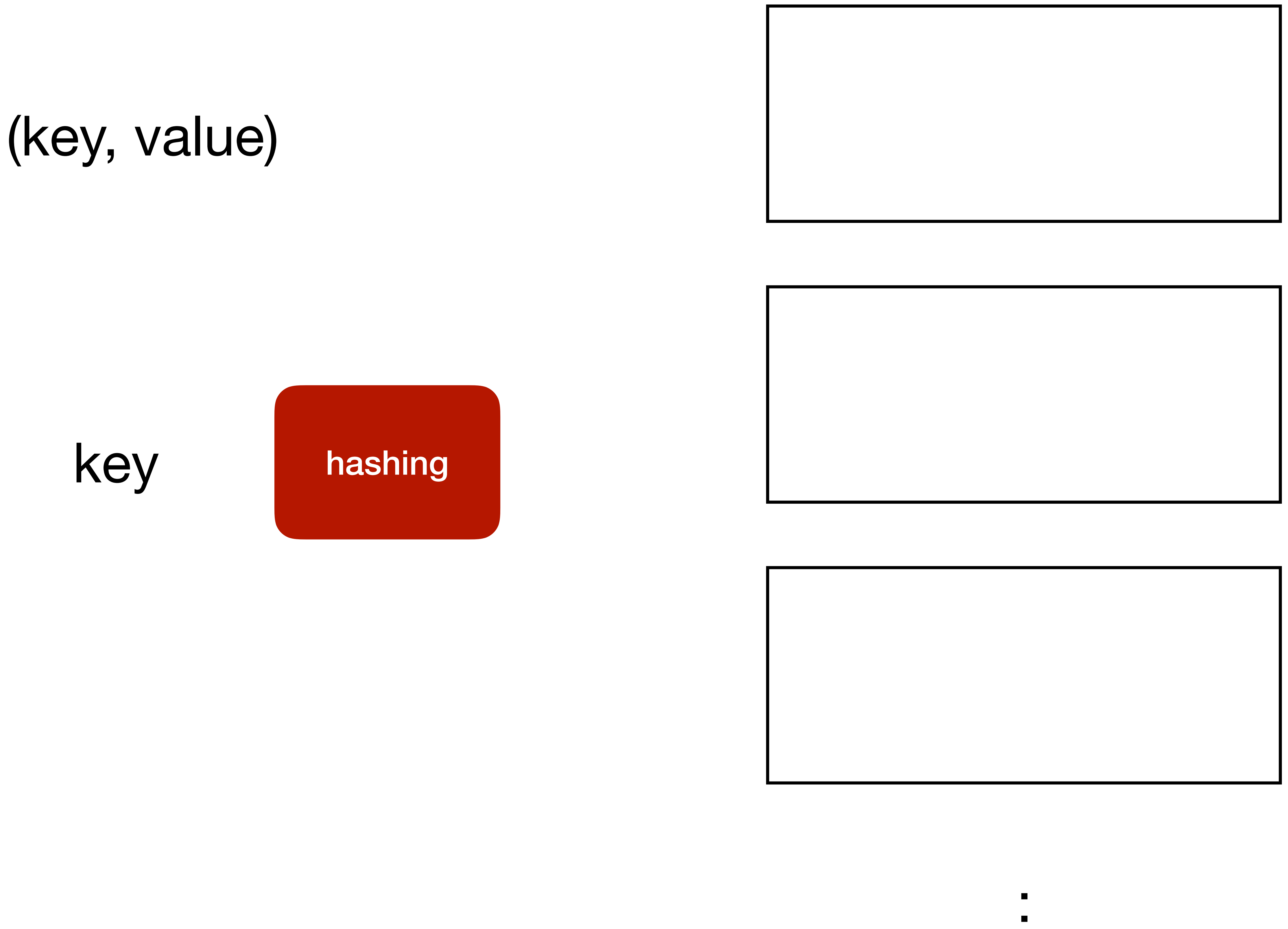

AbstractList<E>
- add(i, e)
- set(i, e)
- get(i)
- remove(i)
- sort(cmp)
ArrayList<E>
-
LinkedList<E>
-


Collection<E>
Set<E>
-
AbstractSet<E>
HashSet<E>


Map<K,V>
- get(Object key)
- put (K key, V value)
- remove(Object key)
- size()
- values()
- entrySet()
- :
AbstractMap<K,V>
HashMap<K,V>

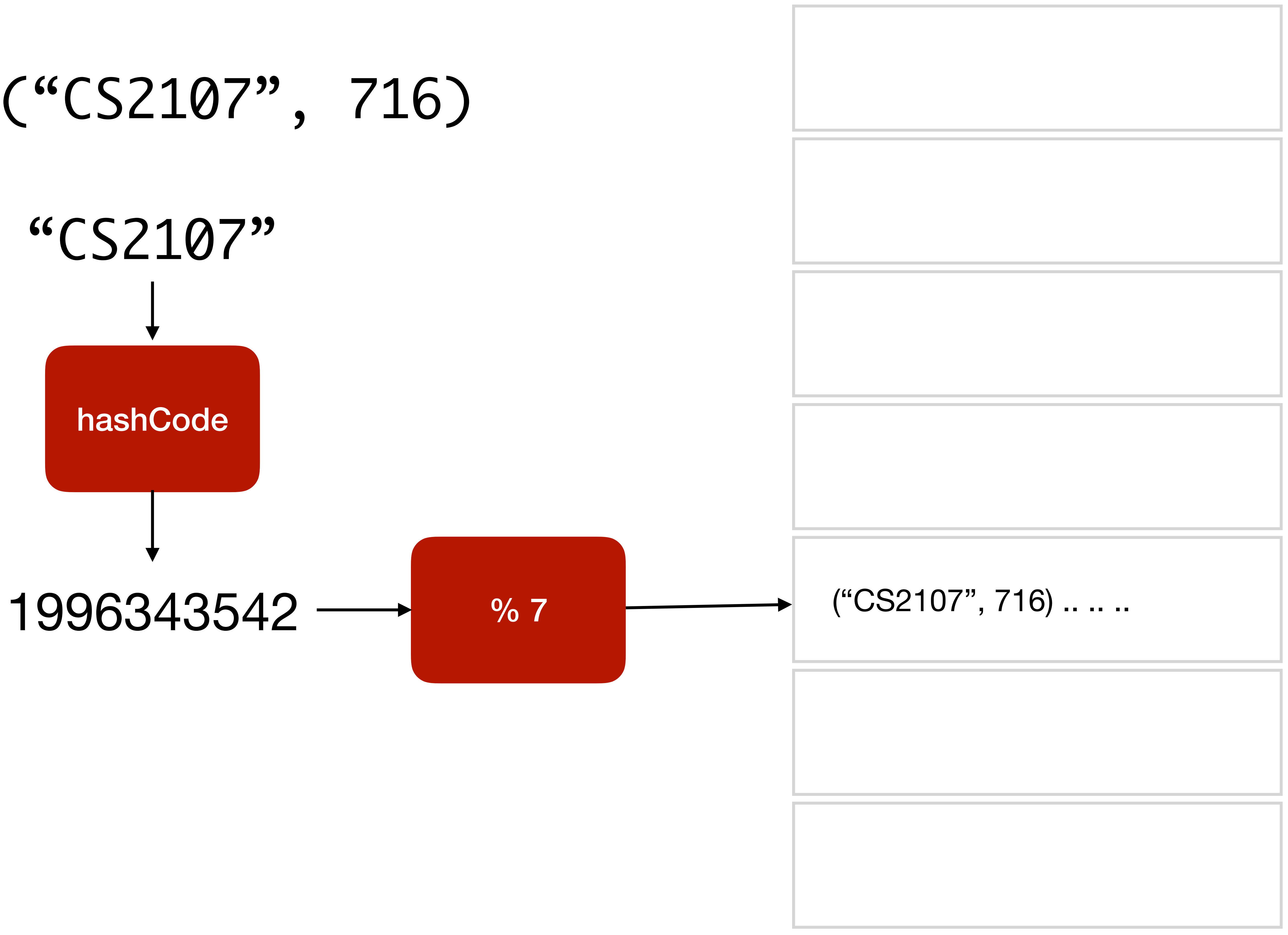
Lecture 6

Hash Code, Nested Class, Enum

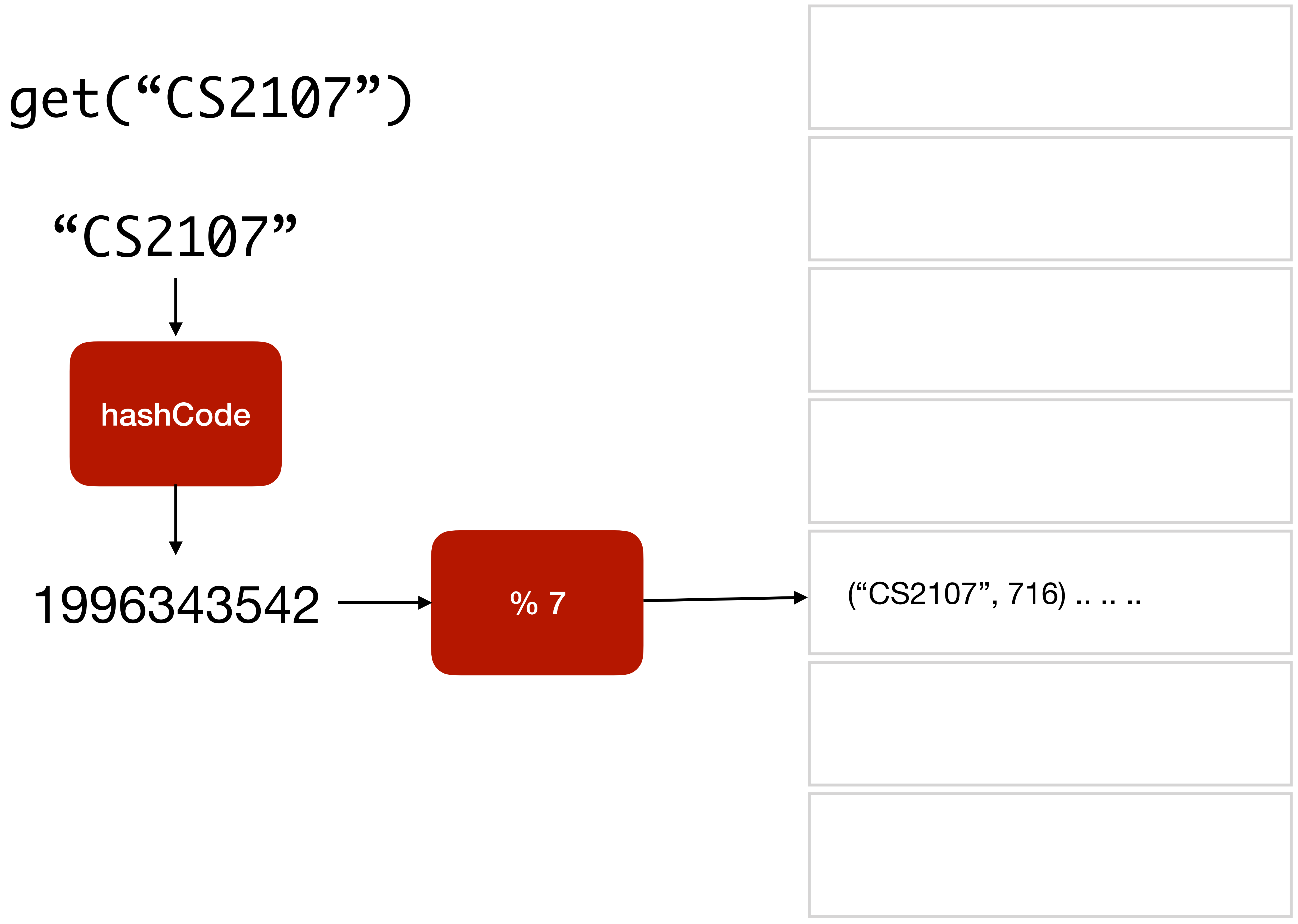

(key, value)
key
hashing
:

e.g., keeping track of bid points for module

```
put(“CS2107”, 716)
```



e.g., keeping track of bid points for module



if

```
  x.equals(y)
```

then

```
  x.hashCode() == y.hashCode()
```

must be true

Nested Class

```
class A {
    :
  static class B {
       :
  }
}
```

Static Nested Class

```
class A {
      :
   class B {
         :
   }
}
```

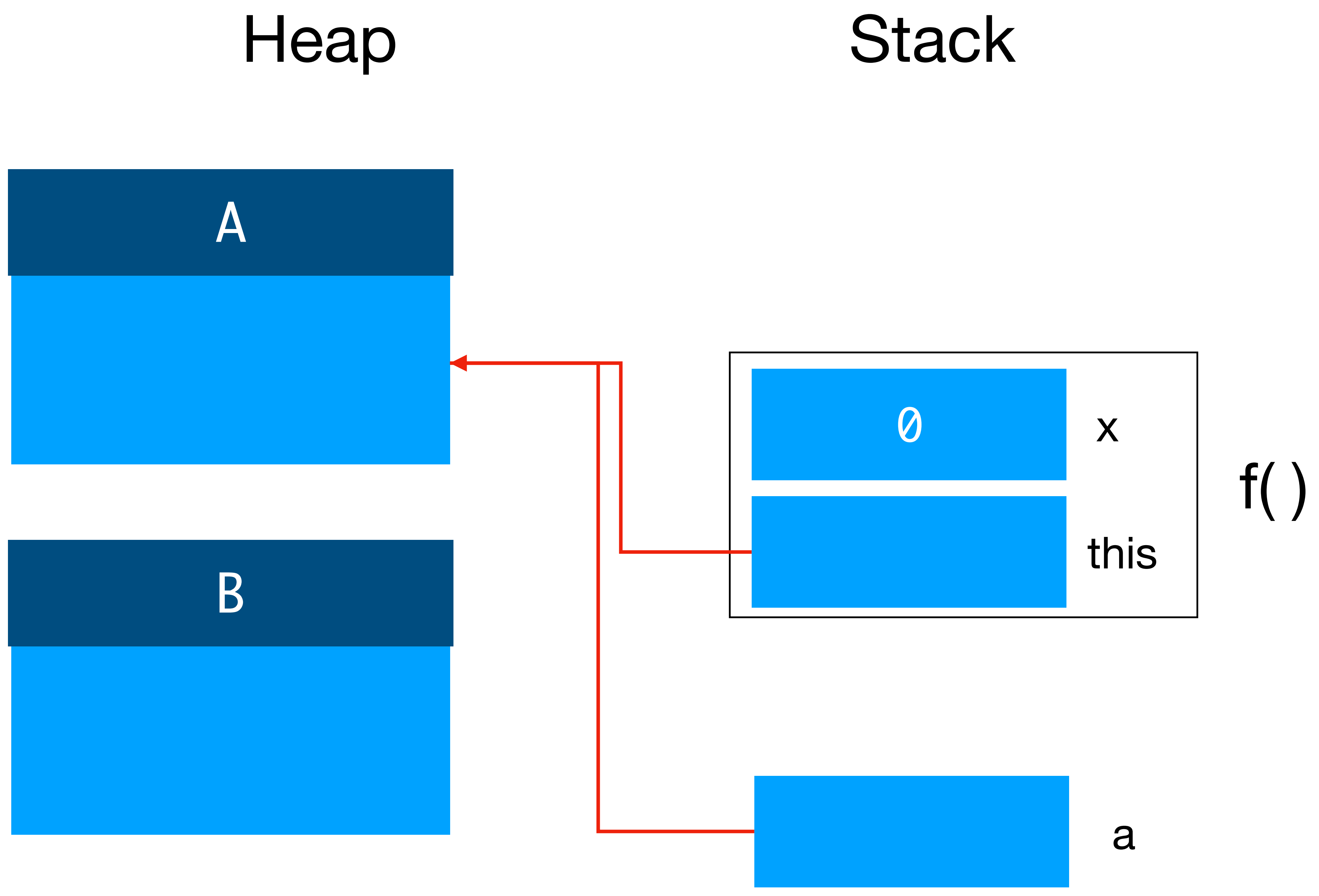
Inner Class

```
class A {
      :
   void f() {
         :
      class B {
            :
      }
   }
```

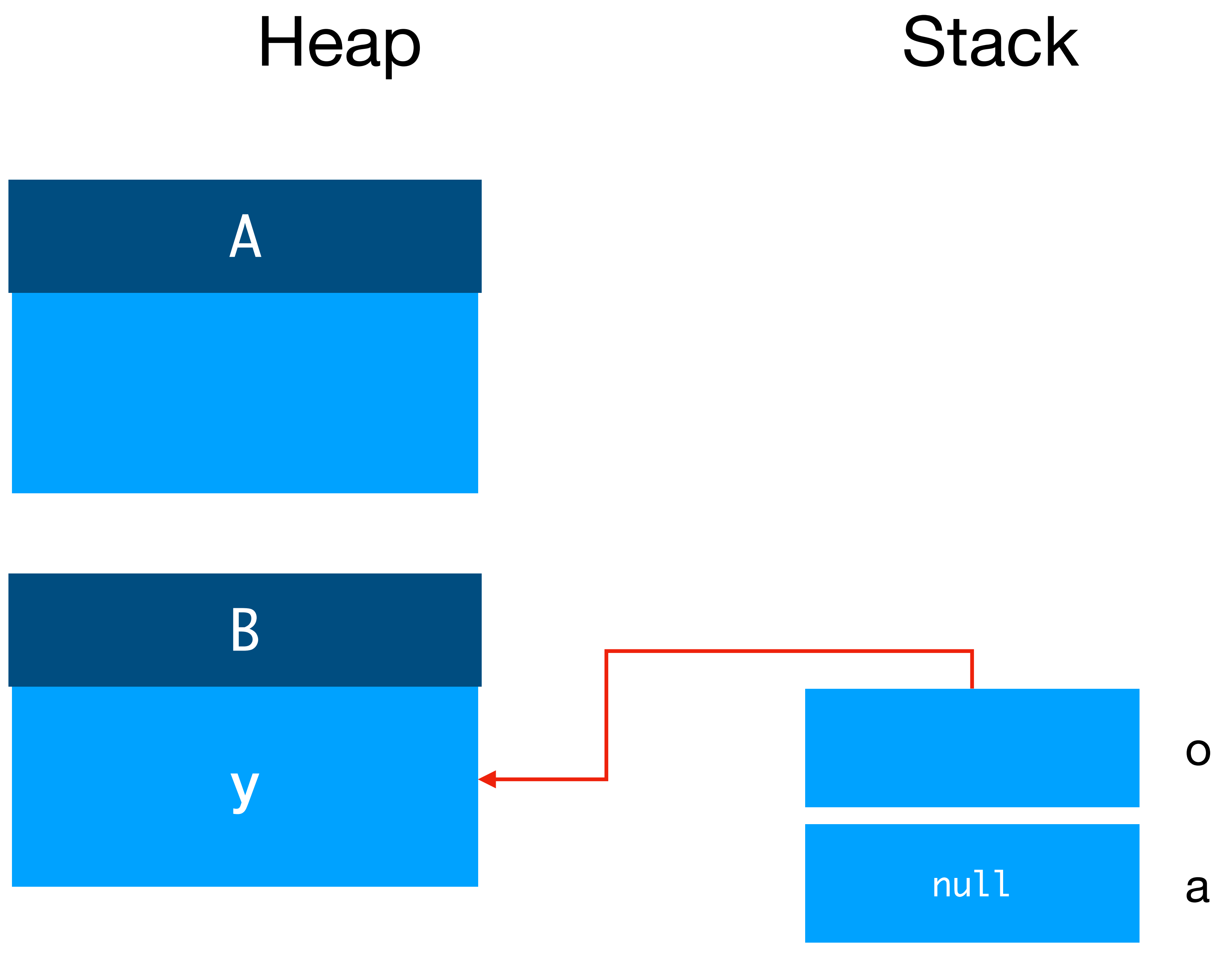
Local Class

```
class A {
  Object f() {
    int x = 0;
    class B {
      // do something with x
    }
    return new B();
  }
}
```

```
A a = new A();
a.f();
```



```
A a = new A();
Object o = a.f();
```



Variable capture: local class makes a copy of local variables used from the enclosing method to within itself

```java
boolean ascendingOrder = true;
class NameComparator implements Comparator<String> {
  public int compare(String s1, String s2) {
    if (ascendingOrder)
      return s1.length() - s2.length();
    else
      return s2.length() - s1.length();
  }
}

ascendingOrder = false;
names.sort(new NameComparator());
```

Java allows capture of `final` or *effectively* final variables only.

```
names.sort(new Comparator<String>() {
  public int compare(String s1, String s2) {
    return s1.length() - s2.length();
  }
});
```

```
new SomeThing(args) {
  body;
}
```

Enumerated Types

```java
public static final int CUSTOMER_ARRIVE = 1;
public static final int CUSTOMER_DONE = 2;
```

```
void foo(int eventType, int customerId) {
  :
}
```

```
foo(customerId, eventType);
```

```
enum EventType {
  CUSTOMER_ARRIVE,
  CUSTOMER_DONE;
}
```

```
void foo(EventType t, Customer c) {
   :
}
```

enum in Java is a class

```
enum EventType {
  CUSTOMER_ARRIVE,
  CUSTOMER_DONE;
}
```

```
public final class EventType extends Enum<EventType>
{
  public static final EventType[] values {..}
  public static EventType valueOf(String n) {..}

  public static final EventType CUSTOMER_ARRIVE;
  public static final EventType CUSTOMER_DONE;
    :
  static {
    CUSTOMER_ARRIVE = new EventType(..);
    CUSTOMER_DONE = new EventType(..);
    :
  }
}
```

```
public abstract class
    Enum<E extends Enum<E>>
    implements ..
```