

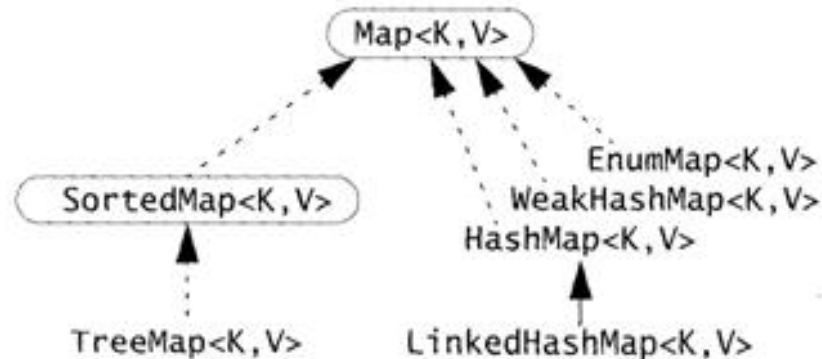
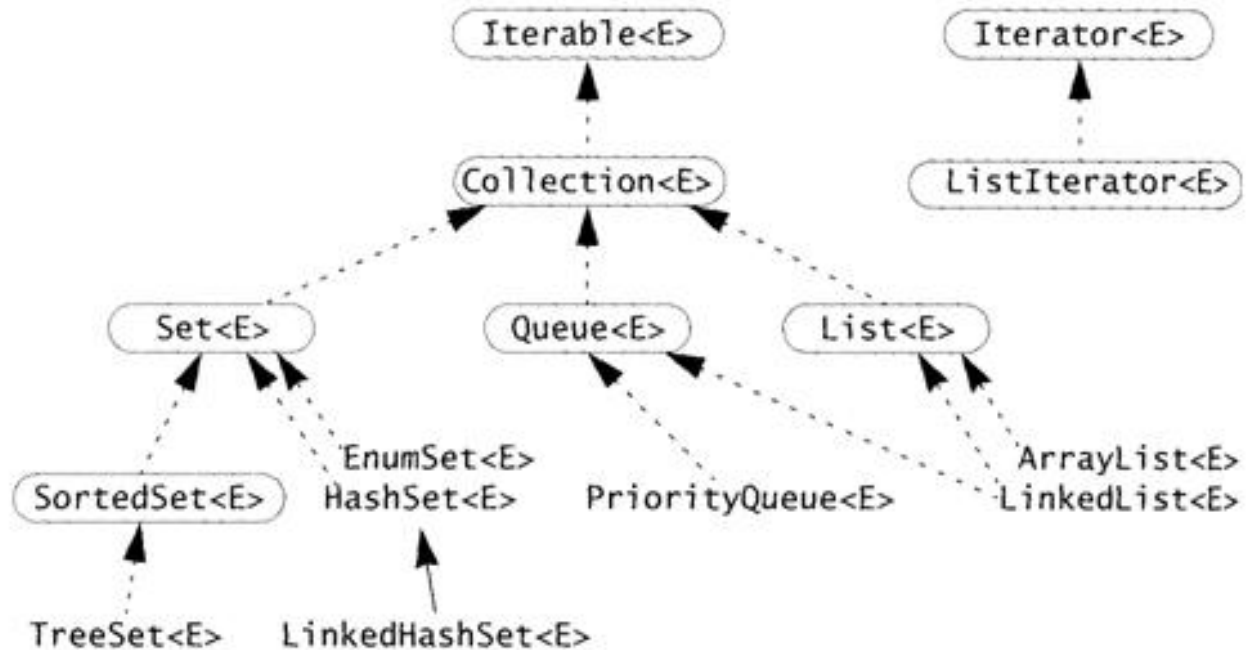
Useful Java Collections

Nathaniel Osgood

MIT 15.879

May 9, 2012

Java.util Collection Hierarchies



Collection Objects in AnyLogic

The screenshot displays the AnyLogic software interface, showing a model diagram for a **Person** agent and the configuration for a **children** collection object.

Model Diagram (Person Agent):

- The diagram shows a state transition model for a **Person** agent.
- States include **Susceptible**, **Infective**, **NonPregnant**, and **Pregnant**.
- Transitions include **InfectionStatechart**, **PregnancyStatus**, and **RandomSex**.
- Actions include **reportLastTimeInfectedForPersons**, **getAndIncrementNextIdForNewPerson**, **dictLastTimeInfectedForPerson**, **nextIdForNewPerson**, **dictLastTimeExposedForPerson**, **getPersonName**, **dictAllTimesInfectedForPerson**, **strName**, **color**, **circlesize**, **CircleSize**, **colorForRelation**, **isInitiallyInfected**, **sex**, **ethnicity**, **mother**, **appearanceTime**, **InitialAge**, **CurrentAge**, **FinalizeDeath**, **reportChildren**, **children**, **FertilityRateAgeSexEthnicity**, **PerformBirth**, **EstablishOffspringConnectionsBasedOnMothersConnections**, **EstablishOffspringLocationBasedOnMothersLocation**, and **causeError**.

Collection Object Configuration (children - Collection):

- Name:** children
- Show name:**
- Ignore:**
- Show at runtime:**
- Access:** public
- Static:**
- Save in snapshot:**
- Collection class:** java.util.ArrayList
- Elements class:** java.util.ArrayList, java.util.LinkedList, java.util.HashSet, java.util.LinkedHashSet, java.util.TreeSet

Informal Collection Variables in AnyLogic

The screenshot displays the AnyLogic software interface for an agent named "Person". The main workspace shows a statechart with three states: "Susceptible", "Infective", and "NonPregnant". Transitions between these states are labeled with "InfectionStatechart". The "Susceptible" state has a transition to "Infective", and the "Infective" state has a transition to "NonPregnant". A "Death" state is also shown, with transitions from both "Susceptible" and "Infective".

The "Person" agent has several variables and methods:

- Variables (V): `color`, `circlesize`, `dictLastTimeInfectedForPerson`, `dictLastTimeExposedForPerson`, `dictAllTimesInfectedForPerson`, `appearanceTime`, `InitialAge`, `CurrentAge`, `sex`, `ethnicity`, `mother`, `FertilityRateAgeSexEthnicity`, `PerformBirth`.
- Methods (F): `reportAllTimesInfectedForPersons`, `saveInfectionHistoryInformation`, `saveExposureAndInfectionMessageInformation`, `reportLastTimeInfectedForPersons`, `getAndIncrementNextIdForNewPerson`, `nextIdForNewPerson`, `getPersonName`, `strName`, `FinalizeDeath`, `reportChildren`.

The "Properties" window is open, showing the configuration for the variable `dictLastTimeInfectedForPerson`:

- Name: `dictLastTimeInfectedForPerson`
- Show name:
- Ignore:
- Show at runtime:
- Access: `public`
- Static:
- Constant:
- Save in snapshot:
- Type: `Other: Hashtable<Person, Double>`
- Initial value: `new Hashtable<Person, Double>()`
- Use Units:
- Unit:

Useful Collections

- Array
- ArrayList
- Linked list
- Dictionary (e.g. implementation: HashTable)
- Set
- Priority queue
- Binary Tree

Common Characteristics

- Capacity to store information
- Iteration thru elements (support for Iterable interface)
- Separation of *interface* from *implementation*
 - There are often several particular “implementations” that can atch a given interface (contract)
 - These differ in the details of “how” they accomplish the tasks prescribed in the implementation, but adhere to the contract
 - You can create your own implementation of an imterface
- Java supports built-in rich set of collections
 - Java.collections
- Many collections use *generics* syntax to allow for “customized versions” for particular contents
 - e.g. ArrayList<Person>, HashMap<String,Person>
 - Note that these “generics” parameters *must be classes!*

Built-in Java Arrays

- Allocated via (explicit or implicit) new operator
- Can optionally list initial contents
- Can contain both “unboxed” (e.g. int) and “boxed” (e.g. Integer) contents
- Syntax Examples
 - `int arrayNeighborIndices[] = new int[n];`
 - `String arrayCities[] = { “Bangor”, “Portland”, “Mooselookmeguntic” };`
- Note that can have array of size 0

Java Arrays: Tradeoffs

- Pros
 - Can easily specify initial contents
 - Simple syntax
 - Boxed & unboxed elements
 - Fast lookup (by index)
- Cons
 - Painful to extend or delete elements (need to manually copy elements)
 - Only integer (int, short) indices

ArrayList

- Generic class
 - Syntax: `ArrayList<Int>`, `ArrayList<String>`
- Pros
 - Rapid insertion & deletion
 - Convenient integer-based indexing
- Note that can have empty `ArrayList`
- Combines good aspects of
 - Array
 - Linked list
- Suggestion: Use a built-in array if you know the size ahead of time

Linked List

- A sequential list of elements of arbitrary length
- Can iterate forward down list
- This is a Doubly Linked List
 - can iterate backwards in list (from end back to beginning)
- Be prepared for potential empty list!
- Application example: Linked List of History Information, *Persons* who have been infected, in order of infection occurrence

Dictionary (Hashtable and HashMap as Implementation)

- If key is the same as value, can be used to implement “content-indexed memory” (and “associative arrays”)
 - Cf:
 - Array: Look up content at integer
 - Dictionary: Can lookup many types of keys
 - e.g. look up information associated with String
- Example use of generics: `HashMap<String,Person>`, `Hashtable<String,ArrayList<Person> >`
- Two associated collections
 - Keys
 - Values
 - (each key can be used to look up an associated value)

Hashtable and HashMap

Implementation of Map

- Pros
 - Rapid insertions (flexible size)
 - Can readily insert items by associated information
- Cons
 - Low bucket count => Risk of clashes between keys => longer time for insertions
 - If too few “buckets”, performance can grow similar to linked list
 - Larger data structure (“wasted space” in the form of empty buckets if load factor is off)
- Application Example: Look up City Characteristics for Names

Hashtable vs. HashMap

- Tolerance for null
 - Hashtable prohibits null keys & values
 - HashMap allows
 - One null key
 - Many null values
- HashMap has a subclass with a predictable order of iteration
- HashMap and Hashtable also differ with respect to multithreading support (beyond scope of course)

Set

- Dichotomous inclusion/omission
 - .contains
- No ordering of elements
 - Cannot tell if A was inserted before B or vice-versa
- Set operations
 - Union (.addAll())
 - Intersection (.retainAll())
 - (Assymetric) Set difference (.removeAll())
- Example: Keeping track of *Persons* that have been infected thus far

Priority Queue

- For a given priority level, this is first-in-first-out
 - First inserted is first to reach “head” of queue
- Can prioritize according to arbitrary comparator
 - Like “first class” vs. “economy” lane, those with higher priority can “skip ahead” of those with lower priority
- A key use lies in representing a waiting list
- Getting element at head
 - Call to `poll()` (returns reference to element and removes from head of queue)
 - Call to `peek()` (returns reference to item at queue head)

Building Your Own Collections

- Java developers routinely create novel “data structures”, including some collections
- Often these data structures are composed of pieces using the above-described (“built-in”) collections
- If you build your own collections, be aware that care should be taken in several areas
 - Need to be careful about passing out references to values from the collection, in case they can be modified
 - Need to be careful about storing away references to external values, since this might allow external code to (typically, inadvertently) modify the data structure internals