



Agent and Object Technology Lab
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma



Ingegneria del software A

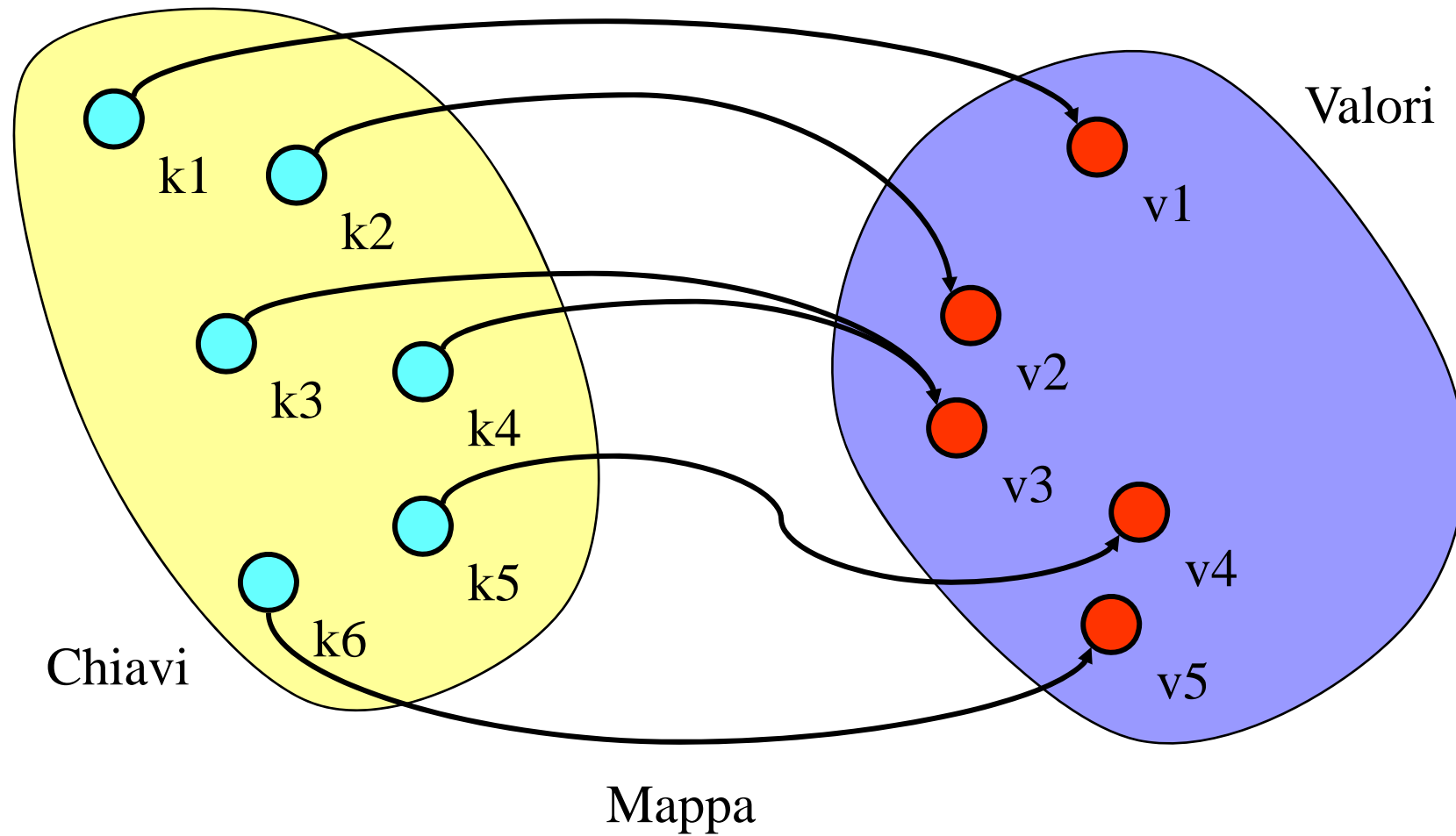
Collezioni di oggetti e tipi generici

Prof. Agostino Poggi

- ◆ Un array permette di gestire insiemi omogenei di dati
 - È un costrutto built-in efficiente che fornisce l'accesso casuale agli elementi e che il tipo degli elementi è fissato a priori (non è necessario il cast)
 - Ma la sua dimensione deve essere fissata a priori
- ◆ Tuttavia in molti casi il numero di dati non è noto a priori e la gestione di questi dati mal si adatta alla struttura dell'array
 - Java fornisce una libreria di classi e interfacce, chiamata, Java Collections Framework, che permette una gestione più flessibile dei dati

- ◆ **Interfacce**
 - Tipi di dati astratti che rappresentano i tipi di collezione
- ◆ **Implementazioni astratte**
 - Parziali implementazioni di interfacce facilmente riadattabili
- ◆ **Implementazioni concrete**
 - Classi basilari che implementano i tipi di collezione fondamentali
- ◆ **Algoritmi**
 - Implementazioni di funzioni basilari come ordinamento e ricerca, applicabili a tutte le collezioni
- ◆ **Utilità**
 - Funzioni basilari applicate alle collezioni e ai loro riferimenti

- ◆ Collection
 - Radice della gerarchia dei diversi tipi di collezione di dati
 - Rappresenta gruppi di oggetti che possono essere (non) duplicati e (non) ordinati
- ◆ List
 - Lista ordinata (o sequenza) di elementi posti in un qualche ordine
- ◆ Set e SortedSet
 - Gruppo di elementi che non contiene duplicati (insieme)
- ◆ Map e SortedMap
 - Gruppo di coppie di oggetti chiave-valore



- ◆ Implementa l'interfaccia `List`
 - Utilizzo di un array per rappresentare la sequenza
- ◆ Ogni istanza di `ArrayList` ha una `capacity`
 - La capacità aumenta automaticamente man mano che gli elementi vengono aggiunti all'oggetto `ArrayList`
 - L'utente può aumentare la capacità
- ◆ Efficienza
 - Aggiungere elementi o ricercare un elemento, senza sapere la sua posizione, richiede un tempo lineare con la dimensione della sequenza

void add(int index, Object element)	Object get(int index)
boolean add(Object o)	int indexOf(Object elem)
boolean addAll(Collection c)	boolean isEmpty()
boolean addAll(int index, Collection c)	int lastIndexOf(Object elem)
void clear()	Object remove(int index)
boolean contains(Object elem)	Object set(int index, Object element)
void ensureCapacity(int minCapacity)	int size()

- ◆ Implementa l'interfaccia `List`
 - Utilizzo di un lista doppiamente concatenata
 - Ogni elemento contiene il riferimento all'elemento successivo e al precedente
- ◆ Efficienza
 - La rimozione di un elemento è molto efficiente
 - È necessario aggiornare solamente i puntatori agli elementi precedenti o successivi all'elemento da rimuovere
 - In una `ArrayList` occorre, invece, spostare tutti gli elementi successivi di una posizione
 - Analogamente l'inserimento di un nuovo elemento comporta solamente l'aggiornamento dei puntatori e non lo spostamento di elementi

void add(int index, Object element)	Object getFirst()
boolean add(Object o)	Object getLast()
boolean addAll(Collection c)	int indexOf(Object elem)
boolean addAll(int index, Collection c)	boolean isEmpty()
void addFirst(Object o)	int lastIndexOf(Object elem)
void addLast(Object o)	Object remove(int index)
void clear()	Object removeFirst()
boolean contains(Object elem)	Object removeLast()
Object get(int index)	int size()

- ◆ ArrayList
 - Ottimizzato l'accesso casuale (basato su array)
 - Non ottimizzati l'inserimento e l'eliminazione all'interno della lista

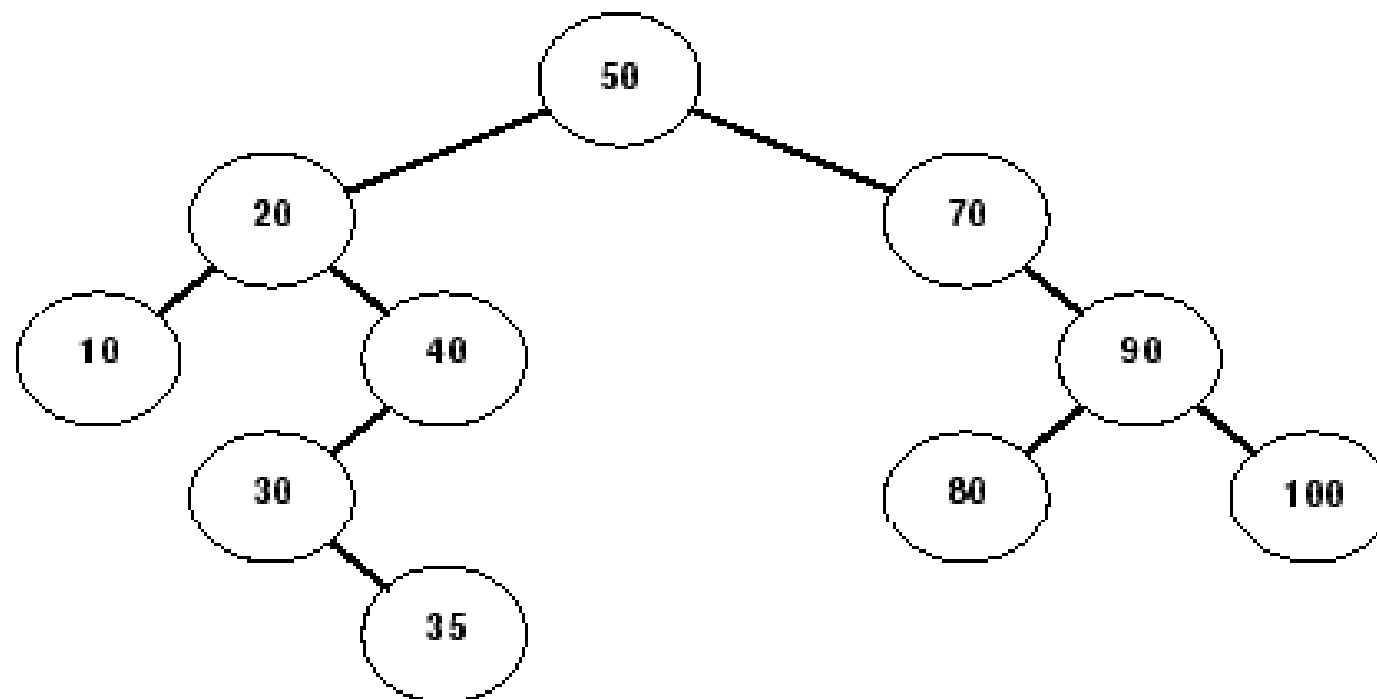
- ◆ LinkedList
 - Ottimizzato l'accesso sequenziale, così come l'inserimento e l'eliminazione
 - Indicato per implementare pile (LIFO) e code (FIFO) infatti fornisce i metodi `addFirst`, `addLast`, `getFirst`, `getLast`, `removeFirst`, `removeLast`

- ◆ Una funzione di hash è una funzione che calcola un numero intero, codice hash, a partire dai dati di un oggetto, in modo che sia molto probabile che oggetti diversi abbiano codici diversi
 - Due o più oggetti possono avere lo stesso codice di hash
 - Questa situazione genera una collisione
 - Una buona funzione di hash deve minimizzare le collisioni

- ◆ Le funzioni di hash vengono usate per creare collezioni (mappe e insiemi) molto efficienti
- ◆ L'idea è quella di avere un array i cui indici siano i codici hash degli elementi
- ◆ Problemi
 - La dimensione dell'array
 - Le collisioni
- ◆ Si deve scegliere una funzione di hash che generi codici in un range ragionevolmente ridotto
- ◆ Si usano liste concatenate

- ◆ Implementa l'interfaccia `Set`
 - Utilizzo di una funzione di hash per accedere e inserire gli elementi
- ◆ Efficienza
 - Se la funzione di hash è ben definita (range piccolo e poche collisioni) le operazioni di verifica di appartenenza, rimozione e inserimento in una collezione hanno costo costante
 - La disuguaglianza tra due oggetti viene controllata inizialmente confrontando il codice hash dei due oggetti e solo in caso di collisione (i due oggetti hanno lo stesso codice hash) viene chiamato il metodo `equals`

- ◆ Un albero binario è un albero in cui ogni nodo ha al massimo due figli
- ◆ Un albero binario di ricerca (ABR) è un albero binario tale che sui valori delle chiavi dei suoi nodi è definito un ordinamento totale per il quale per ogni nodo n dell'albero:
 - Tutte le chiavi dei nodi contenuti nel sottoalbero sinistro di n hanno valore strettamente minore della chiave contenuta in n
 - Tutte le chiavi dei nodi contenuti nel sottoalbero destro di n hanno valore strettamente maggiore della chiave contenuta in n



- ◆ Implementa l'interfaccia `SortedSet`
 - Utilizzo degli alberi binari di ricerca per accedere e inserire gli elementi

- ◆ Questa implementazione garantisce (oltre all'assenza di duplicati) l'ordinamento degli elementi
 - Secondo l'ordinamento naturale, oppure
 - Secondo un ordinamento stabilito da un comparatore definito al momento della creazione dell'insieme (attraverso un opportuno costruttore)


```
class AgeComparator implements Comparator{
    public int compare(Object o1, Object o2){
        int emp1Age = ((Employee) o1).getAge();
        int emp2Age = ((Employee) 22).getAge();

        if (emp1Age > emp2Age)
            return 1;
        else if (emp1Age < emp2Age)
            return -1;
        else
            return 0;
    }
}
```

boolean add(Object o)	boolean isEmpty() int
boolean clear()	Object remove(Object o)
boolean contains(Object o)	Int size()

- ◆ Se è utile l'ordinamento degli elementi, allora bisogna utilizzare la class `TreeSet`, altrimenti è più efficiente la classe `HashSet`
- ◆ `HashSet`: qualora il numero degli elementi presenti nella tabella cresca eccessivamente la struttura deve essere ricostruita su un array di dimensioni maggiori
- ◆ `TreeSet`: il costo dell'inserzione della ricerca e della cancellazione è dell'ordine del logaritmo del numero di chiavi presenti

- ◆ Implementano rispettivamente l'interfaccia `Map` e `SortedMap`
- ◆ Usano rispettivamente il codice hash e gli alberi binari di ricerca per accedere e inserire le chiavi
- ◆ I limiti delle due classi sono simili a quelli riscontrabili nelle classi `HashSet` e `TreeSet`

void clear()	Set keySet()
boolean containsKey(Object key)	Object put(Object key, Object value)
boolean containsValue(Object value)	Object remove(Object key)
Object get(Object key)	int size()
boolean isEmpty()	Collection values()

- ♦ La programmazione generica riguarda la creazione di costrutti di programmazione che possano essere utilizzati con tipi di dati diversi
- ♦ Utilizzando la programmazione generica è possibile creare delle classi generiche parametrizzate rispetto a dei tipi di dati diversi
- ♦ Il tipo con cui in Java si può parametrizzare una classe deve essere un'altra classe e non un tipo primitivo

- ◆ Le classi e le interfacce definite nel Java Collections Framework sono delle classi e interfacce generiche
- ◆ Questa proprietà permette l'istanziamento di collezioni di dati omogenei
 - La consistenza dei dati è controllata durante la compilazione
 - Non è necessario nessun cast

- ◆ È un oggetto che rappresenta un cursore con il quale scandire una collezione a cui è associato
- ◆ Ogni classe che implementa una collezione di dati ha un metodo, `iterator`, che ritorna l'iteratore sulla collezione

```
Collection<E> c = ...  
  
...  
  
Iterator<E> it = c.iterator();  
  
while (it.hasNext()) {  
    E e = it.next();  
    if (!e.consistent())  
        it.remove();  
}
```