UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2019/2020

Strutture dati in java

Collections

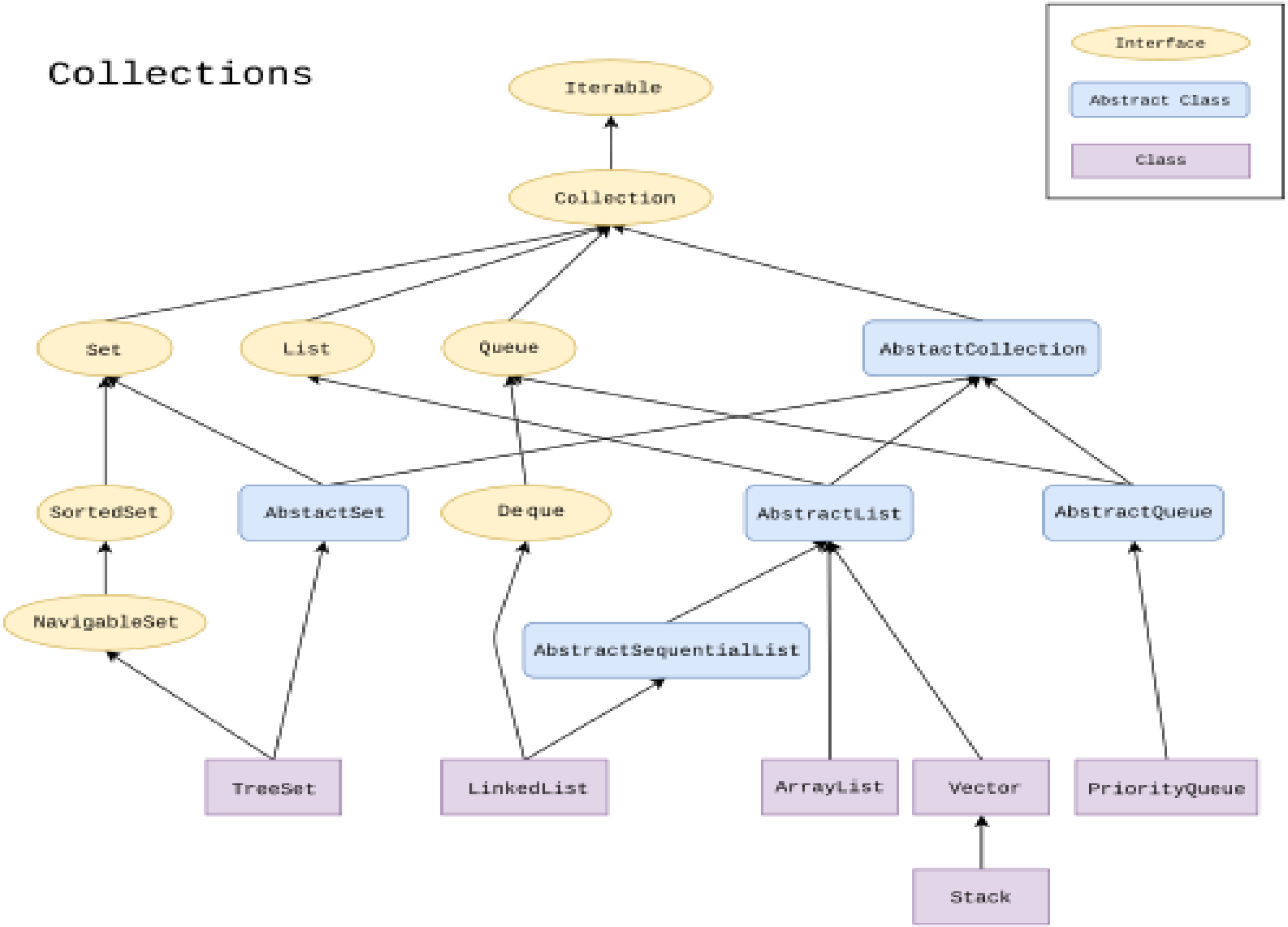
- Una collezione è costituita da un gruppo di oggetti che sono tra loro collegati
- Una collezione è anche definita utilizzando il termine contenitore
- All'interno delle collezioni è possibile effettuare ricerche, aggiungere/rimuovere elementi ecc.
- Java fornisce una libreria/framework per la gestione delle strutture dati più comuni

Strutture dati in java

Collections

- Vector
- List
- Stack
- Queue
- Set

Una visione d'insieme: Collections



Fonte: wikipedia.en

I metodi principali

Metodi

boolean add(T t)

void clear()

boolean contains(Object o)

boolean isEmpty()

boolean remove(Object o)

int size()

Vector

La classe Vector

- I Vector permettono di accedere in una specifica posizione usando l'indice come gli array (metodo get)
- Attenzione: non si può usare l'operatore [] come in C++
- A differenza degli array, la dimensione di un Vector può aumentare o diminuire in base alle esigenze

List

L'interfaccia List

- Rappresenta una sequenza di elementi
- Dà il controllo preciso su dove ogni elemento va inserito
- Permette di accedere agli elementi usando un indice

Implementazioni di List: ArrayList e LinkedList

- ArrayList
  - L'implementazione è molto simile a Vector
  - Accedere ad un elemento usando un indice è veloce
- LinkedList
  - Rappresenta un'implementazione di una lista doppiamente concatenata
  - Accedere ad un elemento usando un indice richiede un'iterazione della lista

ArrayList in pratica

File Persona.java

```
public class Persona {
    private String cf;
    public Persona(String cf) { this.cf = cf; }
}
```

File ProvaLista.java

```
import java.util.ArrayList;
public class ProvaLista {
    public static void main(String[] args) {
        ArrayList<Persona> persone = new ArrayList<Persona>();
        Persona p1 = new Persona("MRARSS80A01H501Z");
        Persona p2 = new Persona("MRARSS80A01H501Z");
        persone.add(p1);
        if(persone.contains(p2))
            System.out.println("Trovato!");
        else
            System.out.println("Non trovato!");
    }
}
```

ArrayList in pratica

Il metodo contains

- Nell'esempio precedente il metodo contains non riesce a capire che p1 e p2 rappresentano la stessa persona

ArrayList in pratica

Il metodo contains

- Nell'esempio precedente il metodo contains non riesce a capire che p1 e p2 rappresentano la stessa persona
- Per implementare il corretto comportamento è necessario ridefinire il metodo equals

```
public boolean equals(Object o) {
  if(o instanceof Persona) {
    Persona p = (Persona) o;
    return cf.equals(p.cf);
  }
  return false;
}
```

Stack/Queue

La classe Stack

- La pila (o stack) rappresenta un gruppo di elementi disposti secondo un criterio LIFO (Last In First Out), in cui l'ultimo elemento inserito è il primo ad essere processato
- Le operazioni principali sono aggiunta, con cui un elemento è aggiunto alla pila, e rimozione, con cui un elemento è rimosso dalla pila

L'interfaccia Queue

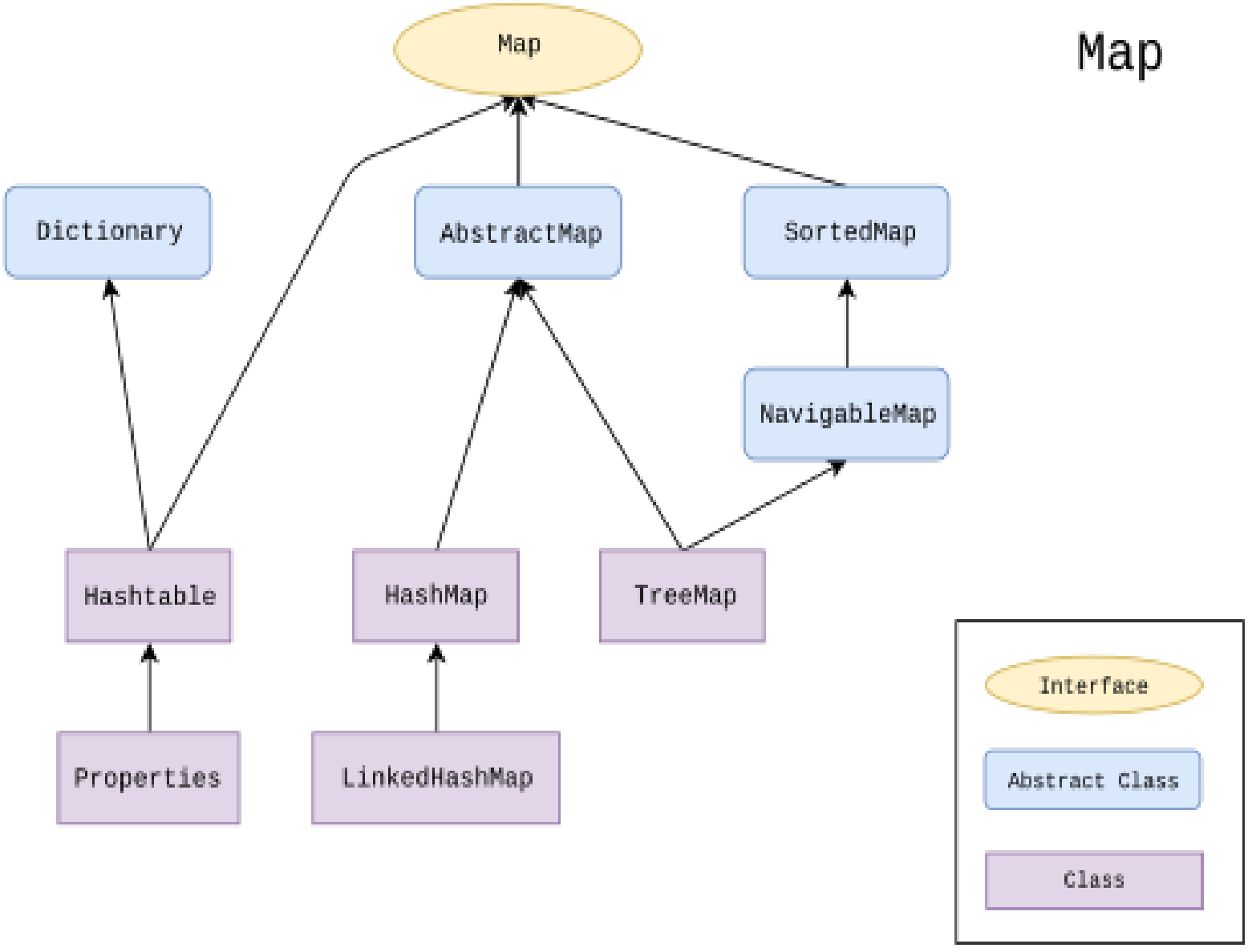
- La coda (o queue) rappresenta un gruppo di elementi disposti secondo un criterio FIFO (First In First Out), in cui il primo elemento inserito è il primo ad essere processato
- Le operazioni principali sono aggiunta, con cui un elemento è aggiunto alla coda, e rimozione, con cui un elemento è rimosso dalla coda
- Un'implementazione concreta è PriorityQueue

Set

L'interfaccia Set

- Set implementa una collection che non contiene duplicati
- Due implementazioni concrete: HashSet e TreeSet
- HashSet: è un'implementazione basata sul concetto di hash, non c'è garanzia che l'ordine degli elementi non cambi
- TreeSet: gli elementi sono ordinati seguendo il loro ordine naturale oppure usando un Comparatore

Una visione d'insieme: Map



Fonte: wikipedia.en

L'interfaccia Map

Mappe

- Un oggetto che collega keys (chiavi) a values (valori)
- Una mappa non può contenere chiavi duplicate
- Esempio: se volessimo associare la marca di un prodotto al numero di prodotti venduti di quella marca, potremmo utilizzare una mappa che collega String e Integer

Implementazioni di Map

- HashMap
- TreeMap

Classe Persona

Riprendiamo la classe Persona dell'esempio precedente, dove abbiamo implementato il metodo equals

File Persona.java

```
public class Persona {
    private String cf;
    public Persona(String cf) { this.cf = cf; }
    public boolean equals(Object o) {
        if(o instanceof Persona) {
            Persona p = (Persona) o;
            return cf.equals(p.cf);
        }
        return false;
    }
}
```

HashMap in pratica

File ProvaHashMap.java

```
import java.util.HashMap;

public class ProvaHashMap {
  public static void main(String[] args) {
    HashMap<Persona,String> m = new HashMap<Persona,String>();
    Persona p1 = new Persona("MRARSS80A01H501Z");
    Persona p2 = new Persona("MRARSS80A01H501Z");
    m.put(p1,"Roma");
    if(m.containsKey(p2))
      System.out.println("Trovato nella map!");
    else
      System.out.println("Non trovato nella map!");
  }
}
```

HashMap in pratica

File ProvaHashMap.java

```
import java.util.HashMap;

public class ProvaHashMap {
  public static void main(String[] args) {
    HashMap<Persona,String> m = new HashMap<Persona,String>();
    Persona p1 = new Persona("MRARSS80A01H501Z");
    Persona p2 = new Persona("MRARSS80A01H501Z");
    m.put(p1,"Roma");
    if(m.containsKey(p2))
      System.out.println("Trovato nella map!");
    else
      System.out.println("Non trovato nella map!");
  }
}
```

In questo caso il metodo containsKey non riesce a capire che p1 e p2 rappresentano la stessa persona, nonostante il metodo equals

HashMap in pratica

Il metodo hashCode()

- Per implementare il corretto funzionamento va ridefinito anche il metodo int hashCode(); dove l' intero restituito è il codice hash per l'oggetto
- Il metodo hashCode() deve rispettare queste regole generali
  1. Quando è eseguito sullo stesso oggetto più volte deve restituire sempre lo stesso intero
  2. Se due oggetti sono uguali eseguendo il metodo equals, allora il metodo hashCode dovrebbe restituire lo stesso intero
  3. Se due oggetti non sono uguali eseguendo il metodo equals, allora hashCode non deve per forza restituire numeri diversi
  4. Comunque è preferibile che due oggetti diversi tra di loro restituiscano interi diversi, questo garantisce performance migliori
  5. Idealmente l'implementazione di hashCode() dovrebbe includere tutti i campi, in modo da produrre risultati diversi

HashMap in pratica

Il metodo hashCode()

- Per implementare il corretto funzionamento va ridefinito anche il metodo int hashCode(); dove l' intero restituito è il codice hash per l'oggetto
- Il metodo hashCode() deve rispettare queste regole generali
  1. Quando è eseguito sullo stesso oggetto più volte deve restituire sempre lo stesso intero
  2. Se due oggetti sono uguali eseguendo il metodo equals, allora il metodo hashCode dovrebbe restituire lo stesso intero
  3. Se due oggetti non sono uguali eseguendo il metodo equals, allora hashCode non deve per forza restituire numeri diversi
  4. Comunque è preferibile che due oggetti diversi tra di loro restituiscano interi diversi, questo garantisce performance migliori
  5. Idealmente l'implementazione di hashCode() dovrebbe includere tutti i campi, in modo da produrre risultati diversi

```
public int hashCode() {
  return cf.hashCode();
}
```