Java

sintassi di base

G. Prencipe
prencipe@di.unipi.it

Basi

- Il nome del file deve essere uguale a quello della classe **public**
  - Al più una classe **public** in ogni file
- L'esecuzione inizia nel metodo **main(String[] args)**
- Usare il **+** per la concatenzazione di **String**he

Array

- Gli array sono definiti e acceduti tramite **[]**
- Gli indici degli array iniziano da **0**
- L'argomento nel **main** è un array di **String**he che contiene gli argomenti passati da linea di comando
  - **args[i]** contiene l'**i**-esimo argomento
- La variabile (di sola lettura) **length** (acceduta tramite il riferimento a un array) restituisce il numero di elementi dell'array
  - **args.length** restituisce il numero di argomenti passati da linea di comando

Creare array

- Dichiarare e allocare un array in un passo

  tipoArr[] nomeArr = {val1, val2, …., valN};

- Esempi

```
int [] valori = {10, 100, 1000};
Point [] punti = {new Point(0, 0),
                  new Point(1, 2)};
```

Creare array

- Dichiarare e allocare un array in due passi
  1. Allocare un array di riferimenti

     ```
     tipoArr [] nomeArr = new tipoArr[dim];
     Point[] punti = new Point[2];
     ```

  2. Riempire l'array

     ```
     punti[0]=new Point(0,0);
     punti[1]=new Point(1,0);
     ```

- Se non si riempie l'array, il valore di default per array numerici è **0**, mentre è **null** per gli array di oggetti

Array multidimensionali

- Sono implementati come array di array

```
int[][] dueD = new int[64][32];
```

Cicli

- **while**

```
while (testBooleano) {
    corpo;
}
```

- **do**

```
do {
    corpo;
} while (testBooleano);
```

- **for**

```
for (init; testBooleano; incr) {
    corpo;
}
```

If

- Opzione singola

```
if (exprBooleana) {
   comando;
}
```

- Opzioni multiple

```
if (exprBooleana) {
   comando1;
} else {
   comando2;
}
```

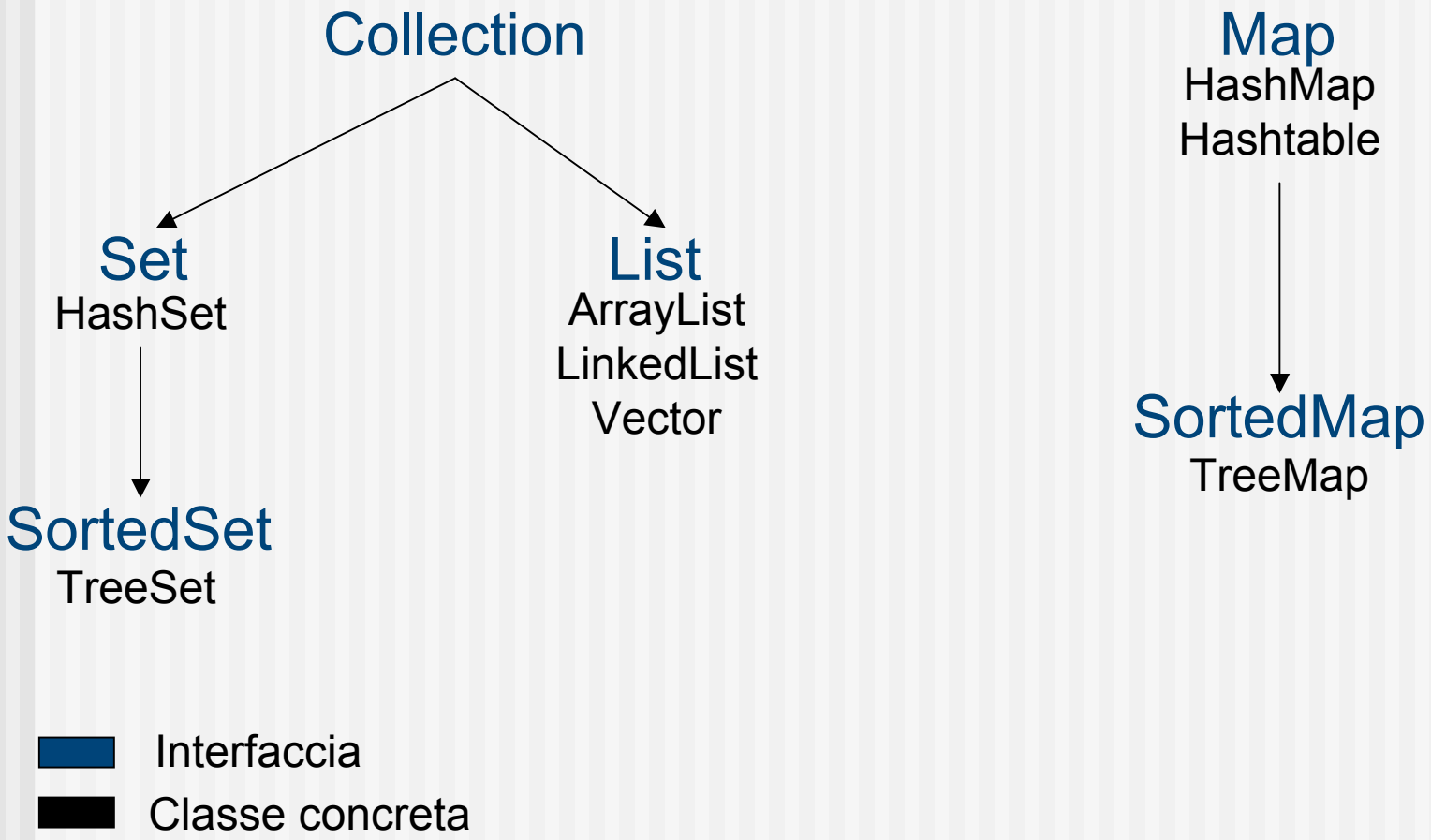
Operatori booleani

- **==, !=**
  - Uguaglianza e disuguaglianza. Ok per i tipi primitivi. Con gli oggetti vengono confrontati i riferimenti e non il contenuto degli oggetti
- **<, <=, >, >=**
  - Minore, minore o uguale, maggiore, maggiore o uguale
- **&&, ||**
  - AND e OR logico. Per entrambi viene usata la valutazione *short-circuit* per calcolare più efficientemente il risultato di espressioni complesse
- **!**
  - Neagazione logica

Stringhe

- **String** è una classe di Java
- Per creare un oggetto di **String** è sufficiente usare le doppie virgolette (“”)
  - Si può utilizzare anche **new** come per la creazione normale di oggetti
- Utilizzare il metodo **equals()** della classe **String** per confrontare **String**he e mai ==

Collezioni


Collection
Set
HashSet
SortedSet
TreeSet
List
ArrayList
LinkedList
Vector
Map
HashMap
Hashtable
SortedMap
TreeMap
Interfaccia
Classe concreta

Interfacce di collezioni

- **Collection**
  - Contenitore di gruppi di oggetti
- **Set**
  - Contenitore di gruppi di oggetti senza duplicati
- **SortedSet**
  - Insieme di oggetti (senza duplicati) memorizzati ordinatamente
  - Necessaria l'implementazione di **Comparable** o **Comparator**

Interfacce di collezioni

- **List**
  - Sequenza (FIFO) di oggetti
- **Map**
  - Memorizza coppie di oggetti **(chiave,valore)**
- **SortedMap**
  - **Map** ordinata sul valore di **chiave**

Classe Collections

- Fornisce una serie di metodi d'utilità per le collezioni
  - **sort**
  - **max**, **min**
  - **reverse**
    - Inverte l'ordine degli elementi nella lista
  - **shuffle**
    - Permuta casualmente l'ordine degli elementi

Classi associate a tipi primitivi

- Ogni tipo primitivo ha una classe associata (detta *Wrapper Class*)
- Il dato primitivo memorizzato nella classe *wrap* è *immodificabile*

Tipo prim	Tipo Wrap
boolean	**Boolean**
char	**Character**
byte	**Byte**
short	**Short**
int	**Integer**
long	**Long**
float	**Float**
Double	**Double**
void	**Void**

Eccezioni

- In Java gli errori sono gestiti tramite eccezioni
- La superclasse di tutte le eccezioni è **Exception** (sottoclasse di **Throwable**)
- Le eccezioni vengono lanciate tramite **throw**
- Un metodo che lancia una eccezione deve
  - Dichiarare nella segnatura che la lancerà, tramite **throws**, oppure
  - Utilizzare un blocco **try-catch** per catturare e gestire l'eccezione
- Non è necessario catturare l'eccezione **RunTimeException**

Il blocco **try-catch**

- Lo schema generico di un blocco **try-catch** è

```
try {
      blocco di comandi;
} catch (TipoEcc1 e1) {
      gestire eccezione e1;
} catch (TipoEcc2 e2) {
      gestire eccezione e2;
} finally {
      codice sempre eseguito;
}
```

La clausola **finally**

- Il codice specificato dalla clausola **finally** viene sempre eseguito, indipendentemente dal lancio di una eccezione

Multithreading

- Tutti i thread sono sottoclasse della classe **Thread** o implementano **Runnable**
- Per codificare correttamente un thread bisogna implementare il metodo **run()**
- Il metodo **run()** serve a mettere in esecuzione un thread
- Viene lanciato non appena viene invocato il metodo **start()**
  - Il metodo **run()** non si invoca direttamente

Il blocco **try-catch**

- Se vengono utilizzate più clausole **catch**, ordinare le clausole dalla più specifica alla più generica
- Se non viene trovata nessuna **catch** che possa gestire l'eccezione, allora l'eccezione viene lanciata dal metodo a un contesto più ampio
  - In questo caso l'eccezione deve essere dichiarata dal metodo con **throws**

Java

sintassi di base

fine