Fondamenti di Java

Static

Modificatori: static

Variabili e metodi associati ad una Classe anziche' ad un Oggetto sono definiti "static".

Le variabili statiche servono come singola variabile condivisa tra le varie istanze

I metodi possono essere richiamati senza creare una istanza.

Variabili "static": esempio 1


```
class  S {
   static int instanceCount = 0; //variabile "di classe"
   S() {instanceCount++;}

}
public class  A {
   public static void main(String a[]) {
         new A();
   }
   A() {
      for (int  i = 0;  i < 10;  ++i) {
         S instance=new S();
      }
      System.out.println("# of instances: "+S.instanceCount);


   }
}
```

```
Output:
# of instances: 10
```

Variabili "static": esempio 2


```
class  S {
   static int instanceCount = 0; //variabile "di classe"
   S() {instanceCount++;}
   public void finalize() {instanceCount--;}
}
public class  A {
   public static void main(String a[]) {
        new A();
   }
   A() {
      for (int  i = 0;  i < 10;  ++i) {
         S instance=new S();
      }
      System.out.println("# of instances: "+S.instanceCount);
      System.gc();
      System.out.println("# of instances: "+S.instanceCount);
   }
}
```

```
Output:
# of instances: 10
# of instances: 0
```

Metodi "static": esempio 1


```
class  S {
   static int instanceCount = 0; //variabile "di classe"
   S() {instanceCount++;}
   static void azzeraContatore() {instanceCount=0;}
}
public class  A {
   public static void main(String a[]) {
        new A();
   }
   A() {
      for (int  i = 0;  i < 10;  ++i) {
         if (i%4==0) S.azzeraContatore();
         S instance=new S();
      }
      System.out.println("instanceCount: "+S.instanceCount);
  }
}
```

Può agire solo su variabili statiche!

```
Output:
instanceCount: 2
```

Ruolo:
Metodi che agiscono su variabili statiche

metodi "static": esempio 2


```
Math.sqrt(double x);
System.gc();
System.arrayCopy(...);
System.exit();
Integer.parseInt(String s);
Float.parseFloat(String s);
```

Notare la maiuscola! (per convenzione)

Ruolo: analogo alle librerie del C

```
Che cos'e':
System.out.println() ?
```

Perchè il main è "static"?


```
public class  A {
    String s="hello";
    public static void main(String a[]) {
        System.out.println(s);
    }
}
```

```
Non static variable s cannot be referenced from static context
```

```
public class  A {
    String s="hello";
    public static void main(String a[]) {
      new A();
    }
    A() {
      System.out.println(s);
    }
}
```

```
hello
```

Sezione: Costruttori

Costruttori

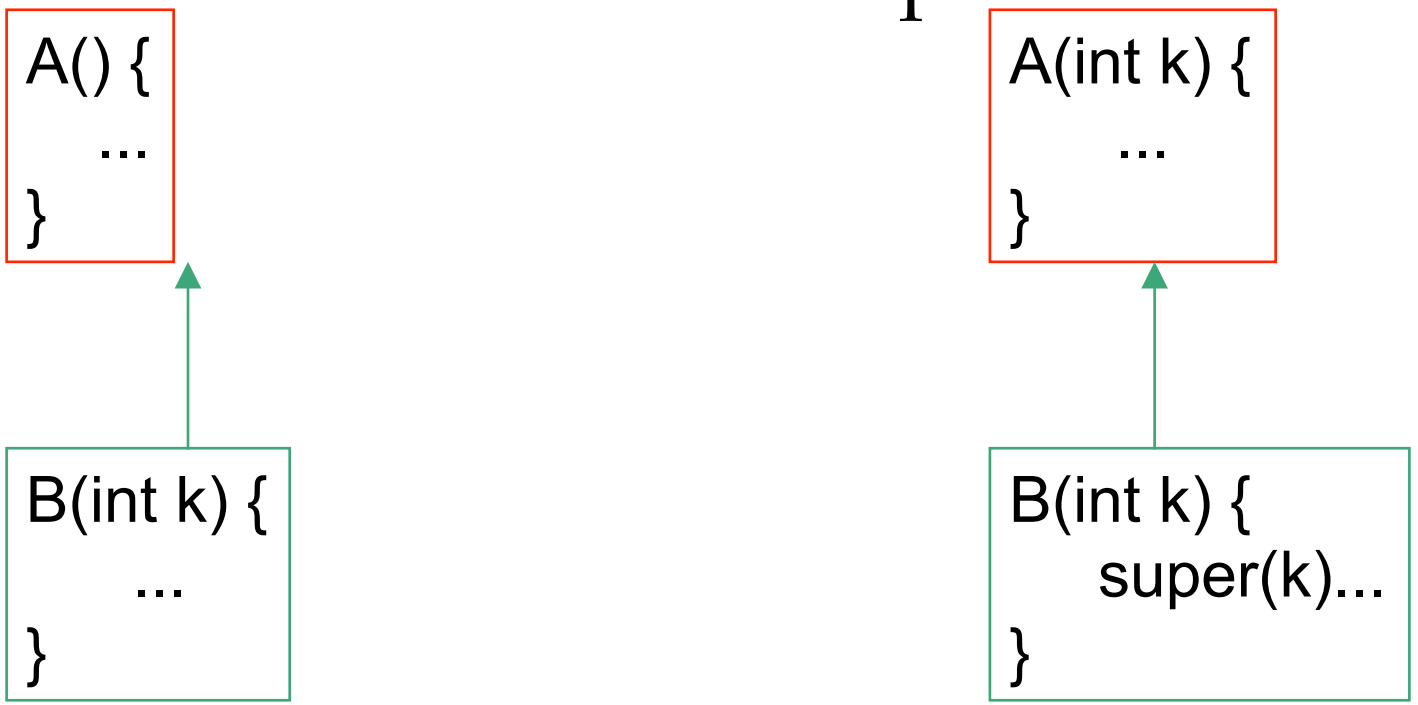
Definizione dei costruttori

Se per una classe A non scrivo nessun costruttore, il sistema automaticamente crea il costruttore A();

Se invece definisco almeno un costruttore non void, ad es. A(int s), il sistema non crea il costruttore A();

Definizione dei costruttori

Se B è figlia di A, il costruttore di B come prima cosa invoca A(), a meno che la prima istruzione non sia una super.



Invocazione dei costruttori


```
public class A {
    public A() {
System.out.println("Creo A");
    }
}
public class B extends A {
    public B() {
System.out.println("Creo B");
    }
    public B(int k) {
System.out.println("Creo B_int");
    }
}
```

Output:
Creo A
Creo B_int

```
public static void main(String [] a) {
    B b=new B(1);
}
```

Invocazione dei costruttori


```
public class A {
 public A(int k) {
System.out.println("Creo A");
                }
               }
public class B extends A {
        public B() {
 System.out.println("Creo B");
                }
        public B(int k) {
System.out.println("Creo B_int”);
                }
               }
```

Output:
ERRORE !

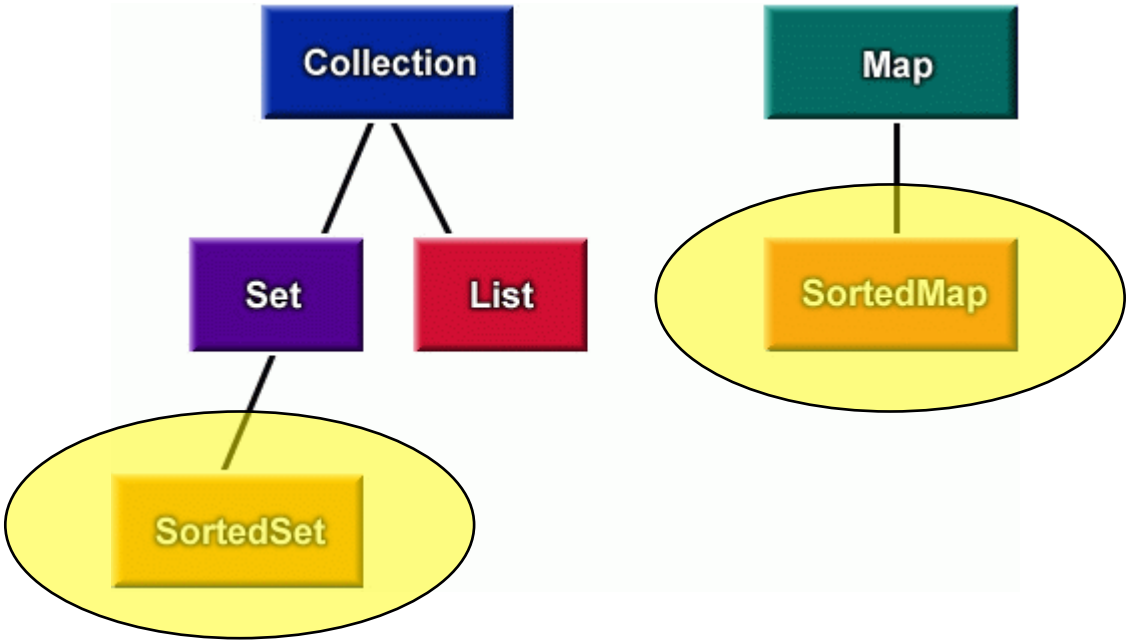
Perchè ?

```
public static void main(String [] a) {
   B b=new B(1);
  }
```

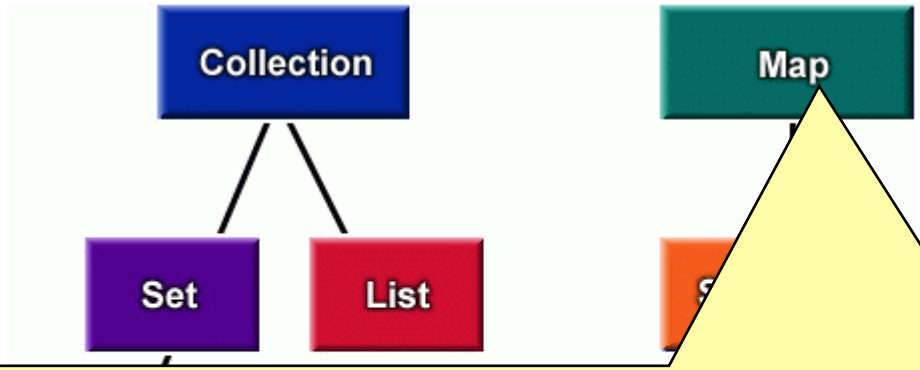
Fondamenti di Java

Collection: approfondimenti

Core Collections Interfaces



Core Collections Interfaces



Map è un entità che abbina oggetti a chiavi di inserimento/ricerca.
Map non può contenere chiavi duplicate: ciascuna chiave è univocamente in relazione con un oggetto.
Esempi di implementazione: tabelle di hash
Esempi concreti: anagrafe, lista clienti…

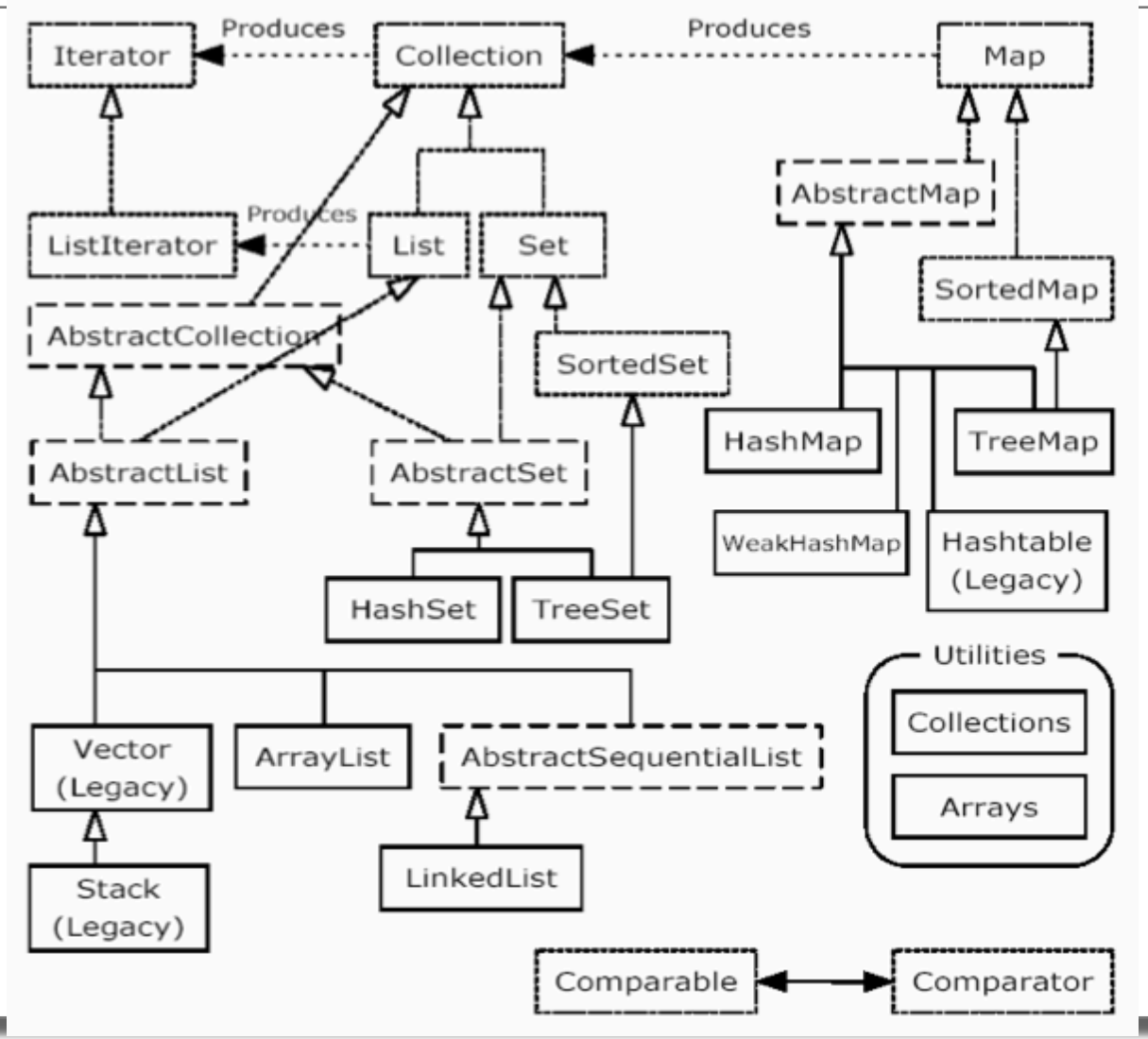
Empty Collections

The Collections class provides three constants, representing the empty Set, the empty List, and the empty Map

Collections.EMPTY_SET
Collections.EMPTY_LIST
Collections.EMPTY_MAP

The main use of these constants is as input to methods that take a Collection of values, when you don't want to provide any values at all.

La tassonomia completa



Fondamenti di Java

Collection: object ordering

Object ordering

Ci sono due modi per ordinare oggetti:

The Comparable interface provides automatic *natural order* on classes that implement it.

The Comparator interface gives the programmer complete control over object ordering. These are *not* core collection interfaces, but underlying infrastructure.

Object ordering with Comparable

A List l may be sorted as follows:

```
Collections.sort(l);
```

If the list consists of String elements, it will be sorted into lexicographic (alphabetical) order.

If it consists of Date elements, it will be sorted into chronological order.

How does Java know how to do this?

String and Date both implement the Comparable interface. The Comparable interfaces provides a *natural ordering* for a class, which allows objects of that class to be sorted automatically.

Comparable Interface

int **compareTo**(**Object** o)

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Definisce l'"ordinamento naturale" per la classe implementante.

Comparable


```
public class Car implements Comparable{

    public int maximumSpeed=0;
    public String name;
    Car(int v,String name) {maximumSpeed=v;
this.name=name;}
    public int compareTo(Object o){
        if (!(o instanceof Car)) {
            System.out.println("Tentativo di
comparare mele e pere!");
            System.exit(1);
        }
        if (maximumSpeed<((Car)o).maximumSpeed)
return -1;
        else return (1);
    }

}
```

Comparable

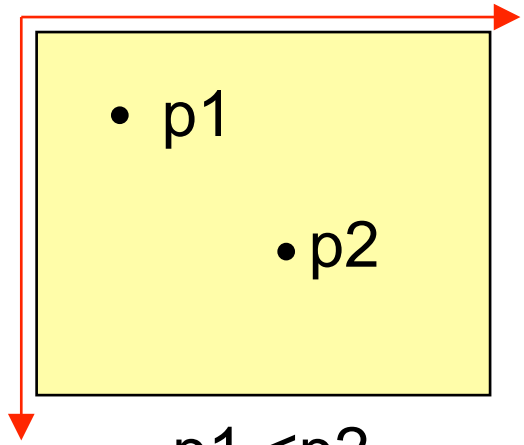

```
class TestCar{
    List macchine=null;
    public static void main(String[] args) {
        new TestCar();
    }
    TestCar(){
        macchine=new LinkedList();
        Car a=new Car(100,"cinquecento");
        macchine.add(a);
        Car b=new Car(250,"porsche carrera");
        macchine.add(b);
        Car c=new Car(180,"renault Megane");
        macchine.add(c);
        printMacchine();
        Collections.sort(macchine);
        printMacchine();

    }
```

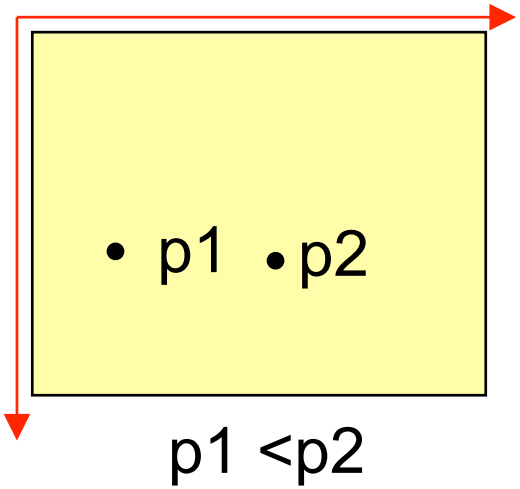
Comparable


```
void printMacchine(){
        Iterator i=macchine.iterator();
        while (i.hasNext()) {
            System.out.println(((Car)i.next()).name);
        }
    }
}
```

Comparable Interface


```
Class Point implements Comparable {
    int x; int y;
    ....
    int compareTo(Object p) {
       … check if Point…
       // ordino sulle y
       retval=y-((Point)p).y;
       // a partità di y ordino sulle x
       if (retval==0) retval=x-((Point)p).x;
       return retval;
}
```





Comparator Interface

int **compare(T o1, T o2)**

Compares its two arguments for order.

```
class NamedPointComparatorByXY
        implements Comparator {
   int compare (NamedPoint p1, NamedPoint p2) {
      // ordino sulle y
      retval=p1.y-p2.y;
      // a partità di y ordino sulle x
      if (retval==0) retval=p1.x-p2.x;
      return retval;
}
```

Comparator Interface


```
class NamedPointComparatorByName
                        implements Comparator {
   int compare (NamedPoint p1, NamedPoint p2) {
       //usa l'ordine lessicografico delle stringhe
       return (p1.getName().compareTo(p2.getName()));
  }
}
... In un metodo di un altra classe:
     // sia c una Collection di NamedPoints
     Comparator cmp1= new NamedPointComparatorByName();
     Comparator cmp2= new NamedPointComparatorByXY();
     List x = new ArrayList(c);
     Collections.sort(x,cmp1)
```


```
import java.util.*;
class EmpComparator implements Comparator {
   public int compare(Object o1, Object o2) {
      EmployeeRecord r1 = (EmployeeRecord) o1;
      EmployeeRecord r2 = (EmployeeRecord) o2;
      return r2.hireDate().compareTo(r1.hireDate());
} }
class EmpSort {
     EmpSort() {
        Collection employees = ... ; // Employee Database
        List emp = new ArrayList(employees);
        Collections.sort(emp, new EmpComparator());
        System.out.println(emp);
     }
     public static void main(String args[ ]) {new EmpSort();}
}
```