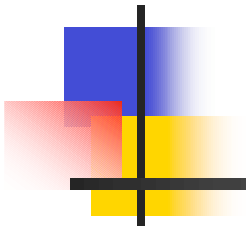
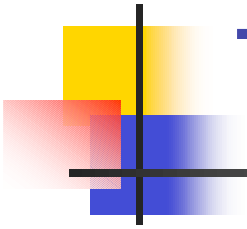


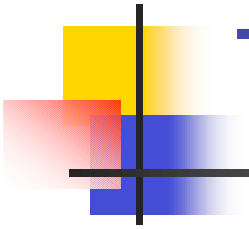
Esercitazione sui BST

3 giugno 2004



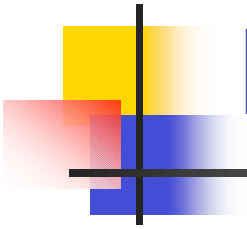
Tipi Wrapper semplici / 1

- I tipi semplici, come int e char, non fanno parte della gerarchia degli oggetti; vengono passati per valore e non possono essere passati per riferimento.
- Alcune classi gestiscono solo oggetti (Vector): abbiamo bisogno di una rappresentazione oggetto dei tipi semplici



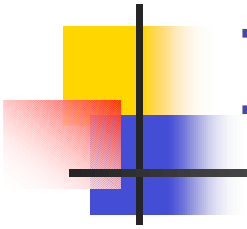
Tipi Wrapper semplici /2

- Java offre classi che corrispondono a ciascuno dei tipi semplici.
- In pratica, queste classi incapsulano (wrap), i tipi semplici all'interno di una classe.
- Tali classi sono definite all'interno di java.lang.



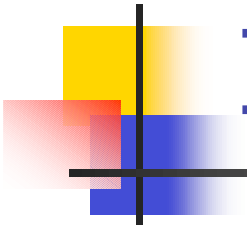
Number

- La classe astratta *Number* è la superclasse delle classi *BigDecimal, BigInteger, Byte, Double, Float, Integer, Long*, and *Short*.
- Le sottoclasse di Number devono fornire metodi per convertire il valore del tipo numerico rappresentato in *byte, double, float, int, long*, e *short*.



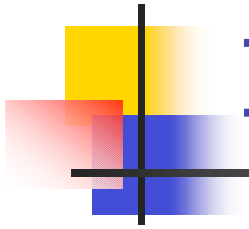
Integer /1

- La classe Integer incapsula il valore del tipo primitivo int in un oggetto. Un oggetto di tipo Integer contiene un unico campo il cui tipo è int.
- Costruttori
  - **Integer**(int value)
  - **Integer**(String s)



Integer /2

- Acuni metodi:
  - byte **byteValue**()
  - int **intValue**();
  - long **longValue**();
  - String **toString**();
- Metodi simili per le classi *Float, Double* e *Long*.
- Esistono wrapper anche per le classi *char* e *boolean*



Il nostro wrapper: BaseObject / 1

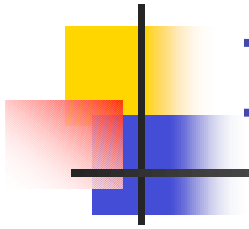
```
// Wrapper class for use with generic data structures.

public class BaseObject implements Comparable {

  private int key;

public BaseObject(int k) {
    this.setKey(k);
   }
public void setKey(int k) {
    key = k;
    }

   public int intValue(){
     return key;
}
```



Il nostro wrapper: BaseObject / 2

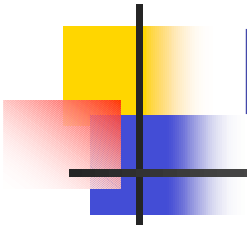
```
public int compareTo(Object bo) {
    if(!(bo instanceof BaseObject))
      throw new ClassCastException();
    if(this.key == ((BaseObject)bo).key)
      return 0;
      else if(this.key < ((BaseObject)bo).key)
        return -1;
            else return 1;
  }

public String toString() {
     return "" + key + " ";
   }
```

Il nostro wrapper: BaseObject / 3

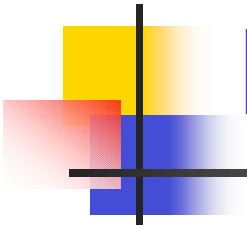
```
public void visit() {
    System.out.print(this.toString());
  }

  /**
   * Implements the equals method.
         * @param anotherBaseObject the second BaseObject.
         * @return true if the objects are equal, false otherwise.
         * @exception ClassCastException if anotherBaseObject is not
         *     a BaseObject.
         */
public boolean equals(BaseObject anotherBaseObject) {
           return this.key == anotherBaseObject.key;
    }
}
```



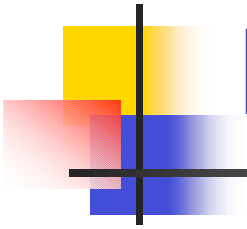
L'interfaccia comparable

- Questa interfaccia impone un ordinamento totale sugli oggetti di ciascuna classe che la implementa.
- Questo ordinamento viene considerato come l'*ordinamento naturale* della classe ed il metodo compareTo definisce il *metodo di confronto*.
- I wrapper di tipi numerici in Java implementano l'interfaccia Comparable.
- Anche BaseObject!



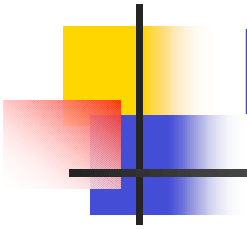
La classe java.util.Random

- La classe *Random* è un generatore di numeri pseudo-casuali, così definiti perché non sono altro che sequenze uniformemente distribuite.
- Definisce i seguenti costruttori:
  - Random(): crea un generatore di numeri casuali. Il suo seme è inizializzato con un valore basato sul tempo corrente: *System.currentTimeMillis()*
  - Random (long seme): crea un generatore di numeri casuali usando il seme dichiarato.
- Metodi:
  - int **nextInt**(int n):Restituisce un valore intero uniformemente distribuito tra 0 (incluso) e lo specificato valore n (escluso), a partire dalla sequenza random corrente.



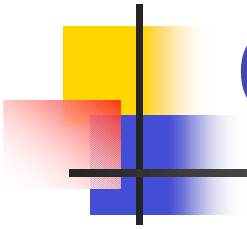
Esercitazione

- Copiare la cartella Esercitazion2 nella propria directory di lavoro
- Studiare le classi date
  - Descritte nelle slide che seguono
- Studiare le classi da completare
  - myBSTree.java
  - BSTTest.java
- Rispondere alle domande



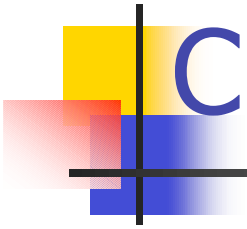
Esercitazione - Classi date

- BaseObject: descritta nelle slide precedenti
- BSTNode: Implementa il generico nodo di un BST
- BSTree: classe che implementa l'ADT BST



Classi da completare - BSTree

- Un'implementazione di BST
- Metodi principali:
  - public Object insert(Comparable el): inserimento nuovo nodo con chiave el
  - public Comparable remove(Comparable el): cancellazione elemento con chiave el
  - public BSTNode search(Comparable el): ricerca elemento con chiave el

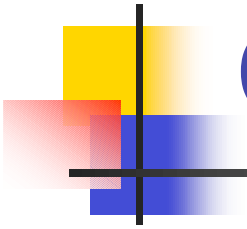


Classi da completare - BSTTest

- E’ una classe che va completata
  - Le funzionalita’ da aggiungere sono descritte da commenti
  - Usa una classe myBSTree che estende BSTree (v. slide seguente)

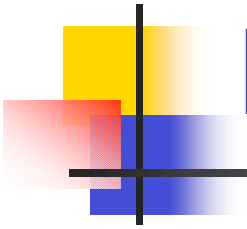
Cosa fare - myBSTree

- Estendere BSTree con nuove funzionalita'
  - public BSTNode lca(BSTNode u, BSTNode v)
    - Trova il LCA dei nodi u e v (v. avanti)
  - public int nrange(BSTNode u, BSTNode v)
    - Dati due nodi u e v, determina no. chiavi comprese nell'intervallo [u.key… v.key] se u.key.compareTo(v.key) <= 0, nell'intervallo [v.key… u.key] se u.key.compareTo(v.key) > 0
  - private int lcaVisit(BSTNode w, Comparable lower, Comparable upper)
    - Dato un nodo w e due chiavi lower e upper, con lower.compareTo(upper) < 0, restituisce le chiavi del sottoalbero avente radice in w che sono comprese nell'intervallo [lower…upper]



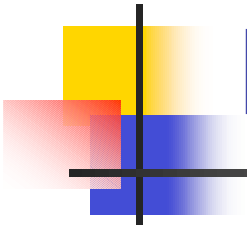
Cosa fare - BSTTest

- Completare l'implementazione del main
  - Si vedano gli esercizi nelle slide seguenti
  - Rispondere a una nuova domanda solo dopo aver verificato la correttezza della risposta alla precedente



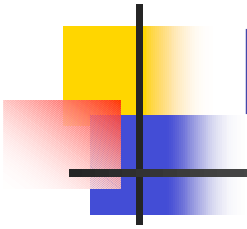
Esercizio 1

- Implementare la parte di BSTTest.java che popola l'albero con un numero NINS di chiavi aventi valori casuali
- Implementare la parte che calcola l'altezza dell'albero risultante
- Studiare l'andamento dell'altezza quando NINS assume valori crescenti che sono potenze di 10 (es. 10, 100, 1000 ecc.)



Esercizio n. 2

- Implementare la funzione public BSTNode lca(BSTNode u, BSTNode v), dove u e v appartengono all'albero
- Il LCA (Least Common Ancestor) è il nodo più profondo che è radice di un sottoalbero che contiene entrambi i nodi
- Implementare la parte di BSTTest.java che inserisce due nodi u e v con chiavi casuali diverse tra loro e ne trova il LCA



Esercizio n. 3

- Implementare il metodo private int lcaVisit(BSTNode w, Comparable lower, Comparable upper) descritto in precedenza
- Usare lcaVisit per implementare il metodo public int nrange(BSTNode u, BSTNode v) descritto in precedenza