Fondamenti di Java

Costruttori
Arrays
Exceptions
UML

Ancora sui costruttori...


```
public class A {
public static void main(String [] ar) {
    A a=new A();
  }
}
```

Corretto:
Viene invocato il costruttore vuoto
A()
sintetizzato dal compilatore

Costruttori sintetizzati


```
public class A {
  A(int k){...}
  public static void main(String [] ar) {
    A a=new A();
  }
}
```

Errore in compilazione:
"A.java": Error #: 300 : constructor A() not found
in class A at line 4, column 8
Viene invocato il costruttore vuoto
A()
Ma il compilatore non lo sintetizza
perchè c'è già un altro costruttore!

Idem...


```
public class B {
}
public class A extends B {
  public static void main
           (String [] ar) {
    A a=new A();
  }
}
```

SI!

```
public class B {
}
public class A extends B {
  A(int k) {...}
  public static void main
           (String [] ar) {
    A a=new A(2);
  }
}
```

SI!

```
public class B {
  B(int k) {...}
}
public class A extends B {
  public static void main
           (String [] ar) {
    A a=new A();
  }
}
```

NO!

Idem...


```
public class B {
  B(int k) {...}
}
public class A extends B {
A(int l) {...}
  public static void main
           (String [] ar) {
    A a=new A(2);
  }
}
```

NO!

```
public class B {
  B(int k) {...}
}
public class A extends B {
A(int l) {super(l);...}
  public static void main
           (String [] ar) {
    A a=new A(2);
  }
}
```

SI!

Arrays

E' possibile definire arrays di tutti i tipi di dati (elementari o classi).

In fase di DICHIARAZIONE non e' necessario specificare la dimensione del vettore.

Solo al momento della ALLOCAZIONE viene richiesto lo spazio desiderato.

```
String[] strings; // oppure String strings[];

strings = new String[10];
strings = new String[20];
```

Arrays of Objects


```
public class A {
  public A() {
    System.out.println("Creo A");
  }
  public static void main(String [] ar) {
    A[] a;
    a=new A[4];
    System.out.println("end");
  }
}
```

Output:
end

Arrays of Objects


```
public class A {
  public A() {
    System.out.println("Creo A");
  }
  public static void main(String [] ar) {
    A[] a;
    a=new A[4];
    for (int k=0; k<4; k++) a[k]=new A();
    System.out.println("end");
  }
}
```

Output:
Creo A
Creo A
Creo A
Creo A
end

Arrays multidimensionali


```
public class A {
  public A() {
    System.out.println("Creo A");
  }
  public static void main(String [] ar) {
    A[][] a;
    a=new A[2][2];
    for (int k=0; k<2; k++)
      for (int j=0;j<2;j++)
        a[k][j]=new A();
  }
```

Output:
Creo A
Creo A
Creo A
Creo A

Arrays multidimensionali – challenge!


```
public class A {
  public A() {
    System.out.println("Creo A");
  }
  public static void main(String [] ar) {
    A[][] a;
    a=new A[2][];
    a[0]=new A[2];
    a[1]=new A[3];
    for (int k=0; k<2; k++)
      for (int j=0;j<3;j++) {
        if (j == 2 && k == 0) continue;
        a[k][j] = new A();
      }
  }
}
```

Qual'e' l'output?

Riuscite a capire la logica delle istruzioni rosse?

Che succede se ometto l'istruzione blu?

Exceptions

Le eccezioni interrompono il normale flusso di un programma per riportare notizia di condizioni eccezionali che possono essere avvenute.

Le eccezioni permettono al programmatore di trattare condizioni inusuali o inattese senza farcire il codice di infiniti controlli di errore.

Ne risulta un codice più facile da leggere e più robusto. Il trattamento di condizioni eccezionali può essere delegato ad un singolo pezzo di codice.

NOTA: anche le eccezioni sono "classi"!

Gestione degli errori


```
public class A {
  public int dividi(int a,int b) {
    int c = 0;
    c=a/b;
    return c;
  }
  public A() {
    int z=dividi(3,0);
    System.out.println("the result is "+z);
  }
  public static void main(String [] ar) {
    A a=new A();
  }
}
```

```
java.lang.ArithmeticException: / by zero
 at pila.A.dividi(A.java:4)
 at pila.A.<init>(A.java:8)
 at pila.A.main(A.java:12)
Exception in thread "main"
```

Exceptions: example 1a


```
public class A {
  public int dividi(int a,int b) {
    int c = 0; //definito qui!
    try {
      c=a/b;
      System.out.println("try block");
    } catch (Exception e) {
      System.out.println("catch block");
      c=Integer.MAX_VALUE;
    } finally {
      System.out.println("finally block");
    }
    System.out.println("and in the end...");
    return c;
  }
```


1b/1

```
public A() {
   int z=dividi(3,2);
   System.out.println("the result is "+z);
 }
 public static void main(String [] ar) {
   A a=new A();
 }
}
```

```
try block
finally block
and in the end...
the result is 1
```

Eccezioni 1b/2

```
public A() {
   int z=dividi(3,2);
   System.out.println("the result is "+z);
 }
 public static void main(String [] ar) {
   A a=new A();
 }
}
```

```
try block
finally block
and in the end...
the result is 1
```

```
public A() {
   int z=dividi(3,0);
   System.out.println("the result is "+z);
 }
 public static void main(String [] ar) {
   A a=new A();
 }
}
```

```
catch block
finally block
and in the end...
the result is 2147483647
```


Errare humanum est..

```
public int dividi(int a,int b) {
   int c=0;
   try {
     c=a/b;
     System.out.println("try block");
   } catch (Exception e) {
     c=a/b;
     System.out.println("catch block ");
   } finally {
     System.out.println("finally block");
   }
   System.out.println("and in the end...");
   return c;
 }
```

Ma che succede se persevero?

finally

L'esecuzione di finally è garantita anche se il catch non temina!

```
public A() {
   int z=dividi(3,0);
   System.out.println("the result is "+z);
 }
 public static void main(String [] ar) {
   A a=new A();
 }
}
```

```
java.lang.ArithmeticException: / by zero
 at pila.A.dividi(A.java:13)
 at pila.A.<init>(A.java:4)
 at pila.A.main(A.java:22)
finally block
Exception in thread "main"
```



```
public int dividi(int a,int b) {
   int c=0;
   try {
     c=a/b;
     System.out.println("try block");
   } catch (Exception e) {
     System.out.println("Errore di tipo "+e.getClass());
     c=Integer.MAX_VALUE;
   } finally {
     System.out.println("finally block");
   }
   System.out.println("and in the end...");
   return c;
 }
```

```
Errore di tipo class java.lang.ArithmeticException
finally block
and in the end...
the result is 2147483647
```

```
public class A {
  public A() {
    try {
      int z = modulo(3, 0);
      z = dividi(3, 0);
    } catch (Exception e) {
      System.out.println("Errore!!");
    }
  }
  public int dividi(int a,int b) throws Exception {
    System.out.println("eseguo dividi");  return a/b;
  }
  public int modulo(int a,int b) throws Exception {
    System.out.println("eseguo modulo");  return a%b;
  }
  public static void main(String [] ar) {A a=new A();}
}
```

```
eseguo modulo
Errore!!
```

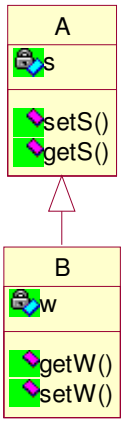

Lo scaricabarile...

```
public class A {
  public A() throws Exception {
      int z = modulo(3, 0);
      z = dividi(3, 0);
  }
  public int dividi(int a,int b) throws Exception {
    System.out.println("eseguo dividi");      return a/b;
  }
  public int modulo(int a,int b) throws Exception {
    System.out.println("eseguo modulo");      return a%b;
  }
  public static void main(String [] ar) throws Exception {
    A a=new A();
  }
}
```

```
java.lang.ArithmeticException: / by zero
  at pila.A.modulo(A.java:15)
  at pila.A.<init>(A.java:3)
  at pila.A.main(A.java:11)
eseguo modulo
Exception in thread "main"
```

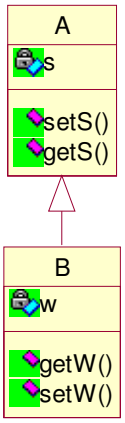
UML – Class Diagram

- rappresenta le classi e gli oggetti che compongono il sistema, ed i relativi attributi ed operazioni
- specifica, mediante le associazioni, i vincoli che legano tra loro le classi
- può essere definito in fasi diverse (analisi, disegno di dettaglio)



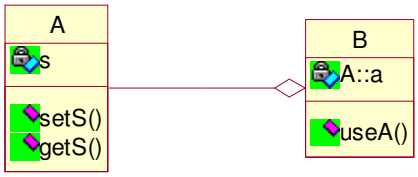
UML: Ereditarietà – "is"


```
class A {
    int s;
    public void setS(int) {…};
    public int getS() {…};
}
class B extends A {
    int w;
    public void setW(int) {…};
    public int getW() {…};
}
```



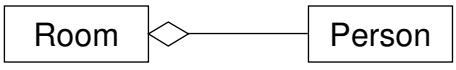
UML: Aggregazione


```
class A {
    int s;
    public void setS(int){…};
    public int getS() {…};
}
class B {A ob;
    public void useA() {…};
}
```

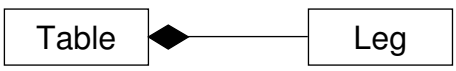


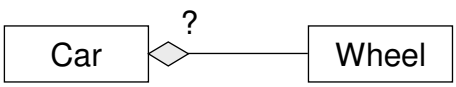
Aggregation - Composition

Use *aggregation (has-a)* when the lifecycle of the partecipating elements is different (one can exist without the other).

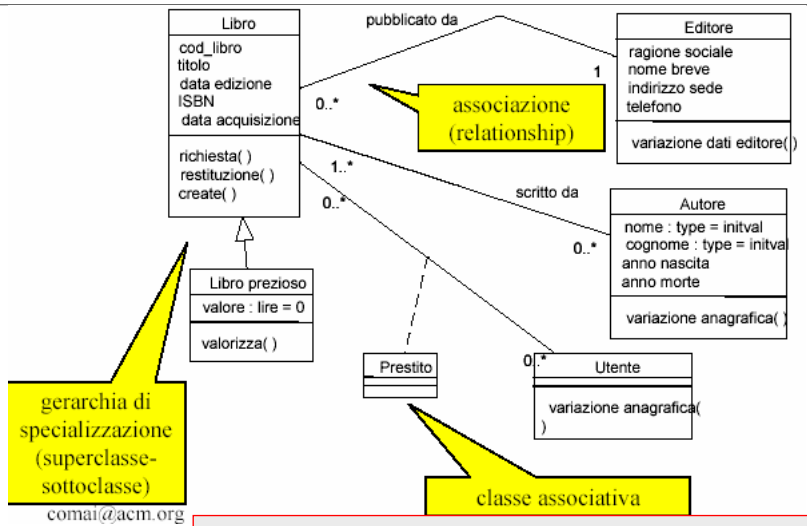


Use *composition (part-of)* when the *container* cannot be conceived without the *contained*.





UML – Class Diagram



comai@acm.org

Fac.Scienze – Università di Trento

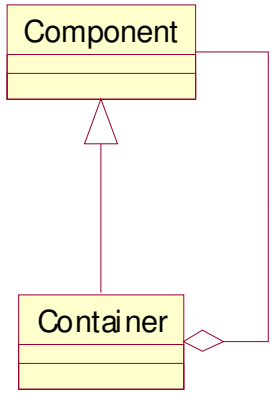
Disegno ripreso da: Adriano Comai
http://www.analisi-disegno.com/a_comai/corsi/sk_uml.htm

UML: component - container

Un Container contiene [0 o +] Components

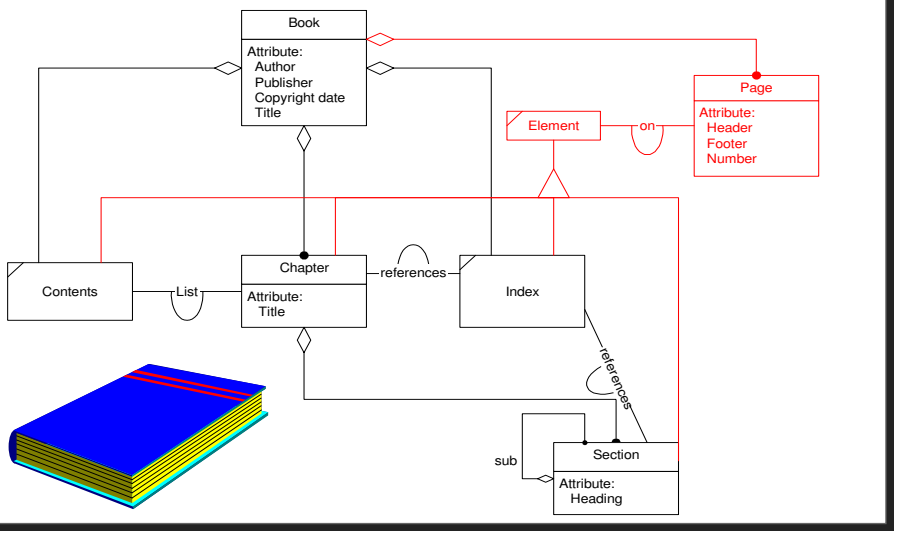
Un Container è un Component

(quindi il contenimento è ricorsivo)



Fac.Scienze – Università di Trento

Object Diagram for a Book



UML: menu hierarchy

MenuComponent è una classe astratta

Ad un Frame può essere associato un Menubar

Un MenuBar contiene dei Menu

Un Menu contiene dei MenuItem

Un Menu è un MenuItem (risorsività del contenimento)