Confronto dell'operatore new

```
in C++:  Point * punto = new Point(10,10);

in Java:  Point  punto = new Point(10,10);
```

punto.x di Java equivale a punto->x del C++

In Java gli oggetti sono accessibili
SOLO per referenza

.

memory management

La gestione (dinamica) della memoria e' automatica, tramite
la creazione (operatore new) e
la distruzione (garbage collection) di oggetti.

GC interviene quando serve memoria.

GC elimina gli oggetti per i quali non vi sono piu' riferimenti attivi.

GC puo' essere attivato su richiesta esplicita: System.gc()

memory management - costruttori

Operazioni da eseguirsi alla nascita di un oggetto vanno definite nel metodo “costruttore”.

Ogni classe deve avere uno (o più) costruttori.

I costruttori possono differire per numero e tipo di parametri.

Es.:

```
Pila() {
    size=100; …
}

Pila(int size) {
    this.size=size
}
```

memory management - distruttori

Operazioni da associarsi con l'eliminazione di un oggetto possono essere definite nel metodo "distruttore" finalize() (opzionale)

NOTA: il metodo finalize POTREBBE NON ESSERE CHIAMATO DAL SISTEMA (es. se il programma finisce prima…)

Per essere certi che vengano chiamati i metodi finalize, occorre chiamare la

System.runFinalization() subito DOPO la System.gc()

System agisce come libreria

System.out.println(...);

System.gc();

System.runFinalization();

System.exit(int status);

System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length);

long System. currentTimeMillis();

Esercizio:
Costruite una Coda analoga alla Pila

Trasformare la Pila in Coda

```
package strutture;
public class Coda{
  int estrai() {
      assert(marker>0):"Invalid marker";
      int retval=contenuto[0];
      for (int k=1; k<marker; k++ )
        contenuto[k-1]=contenuto[k];
      marker--;
      return retval;
    } // il resto è uguale…
```

```
int estrai() {//la estrai di Pila
  assert(marker>0):"Invalid marker";
  return contenuto[--marker];
}
```

Ereditierietà

La estensioni possono essere:

STRUTTURALI

(aggiunta di varabili di istanza)

e/o

COMPORTAMENTALI

(aggiunta di nuovi metodi

e/o

modifica di metodi esistenti)

Trasformare la Pila in Coda

```
package strutture;
public class Coda extends Pila{
  int estrai() {
      assert(marker>0):"Invalid marker";
      int retval=contenuto[0];
      for (int k=1; k<marker; k++ )
        contenuto[k-1]=contenuto[k];
      marker--;
      return retval;
    }
```

```
int estrai() {//la estrai di Pila
  assert(marker>0):"Invalid marker";
  return contenuto[--marker];
}
```

Trasformare la Pila in Coda

```
public static void main(String args[]) {
        int dim=5;
        Coda s=new Coda();
        for (int k=0;k<2*dim;k++)
             s.inserisci(k);
        for (int k=0;k<3*dim;k++)
           System.out.println(s.estrai());
    }
}
```

subclassing & overriding

```
public class Point {
 public int x=0;
 public int y=0;
 Point(int x,int y){
   this.x=x;
   this.y=y;
 }
 public String toString(){
    return "("+x+","+y+")";
 }
 public static void main(String a[]){
    Point p=new Point(5,3);
    System.out.println(p);
  }
}
```

Output:
(5,3)

subclassing & overriding

```
public class NamedPoint extends Point {
  String name;
  public NamedPoint(int x,int y,String name) {
    super(x,y); //prima istruzione!
    this.name=name;
  }
  public String toString(){ //Overriding
    return name+" ("+x+","+y+")";
  }
  public static void main(String a[]){
    NamedPoint p=new NamedPoint(5,3,"z");
    System.out.println(p);
  }
}
```

Output:
z (5,3)

subclassing & overriding

```
public class NamedPoint extends Point {
  String name;
  public NamedPoint(int x,int y,String name) {
    super(x,y); //prima istruzione!
    this.name=name;
  }
  public String toString(){ //Overriding
    return name+super.toString();
  }
  public static void main(String a[]){
    NamedPoint p=new NamedPoint(5,3,"z");
    System.out.println(p);
  }
}
```

Output:
z (5,3)

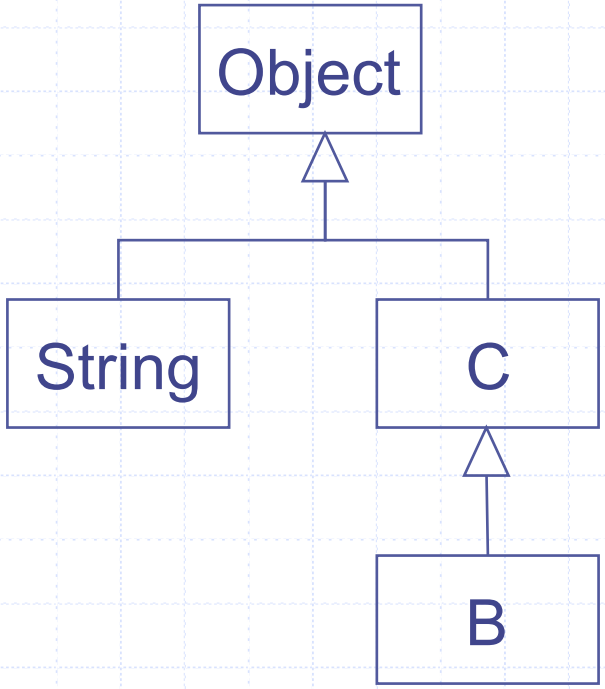
Overloading - Overriding

Overloading:
Funzioni con uguale nome e diversa firma possono coesistere.
move(int dx, int dy)
move(int dx, int dy, int dz)

Overriding:
Ridefinizione di una funzione in una sottoclasse (mantenendo immutata la firma)
Es. estrai() in Coda e Pila

Unified Modeling Language

Esiste una notazione grafica per mostrare le relazioni di ereditarietà.

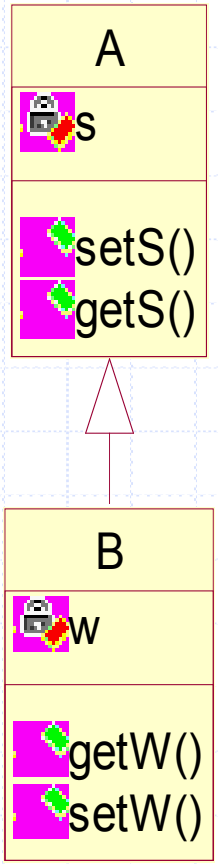



```
class C {…}
class B extends C
{…}
```

Tutte le classi ereditano da Object!

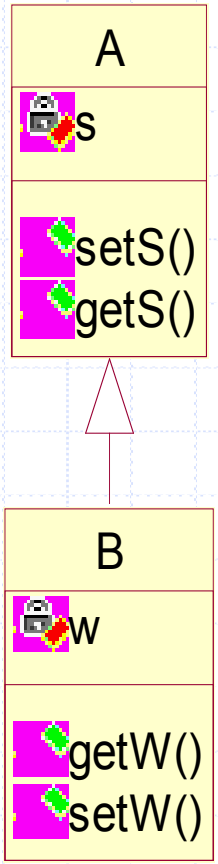
UML – Class Diagram

- 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)



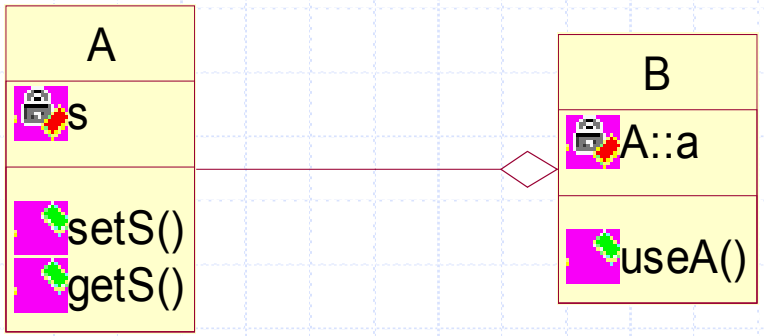
UML: Ereditarietà – “is”

```
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

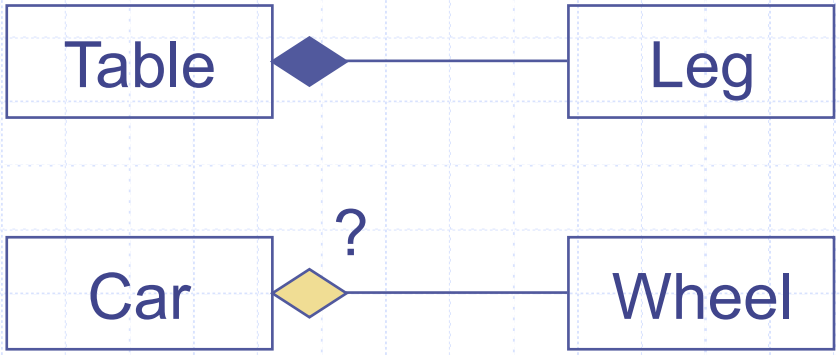
```
class A {
      int s;
      public void setS(int){…};
      public int getS() {…};
}
class B {A ob;
      public void useA() {…};
}
```



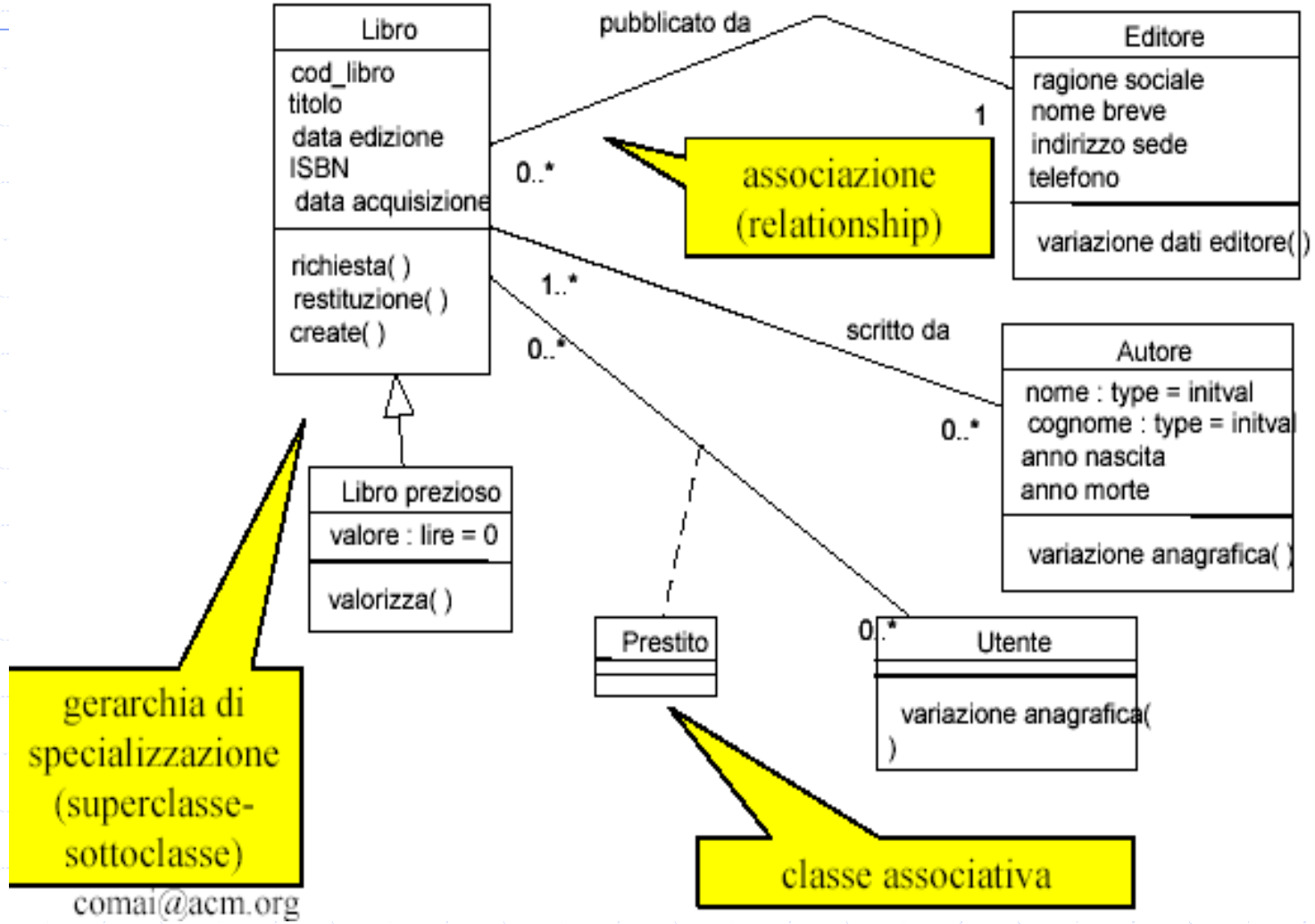
Aggregation - Composition

- 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*.

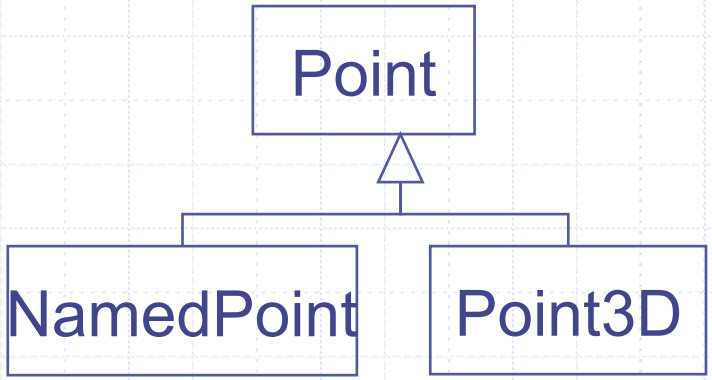


UML – Class Diagram



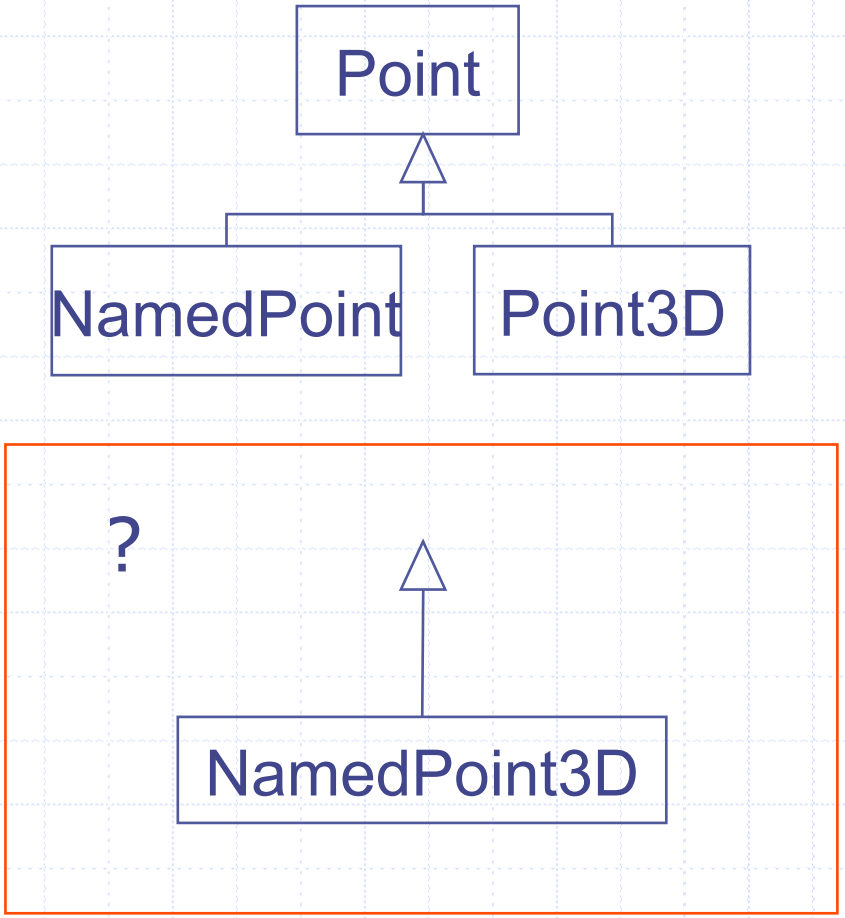
Disegno ripreso da: Adriano Comai
http://www.analisi-disegno.com/a_comai/corsi/sk_uml.htm

Esercizio



- a) Scrivere un metodo move(int dx, int dy) in Point.
- b) Estendere Point a Point3d aggiungendo una coordinata z, e fornendo un metodo move(int dx, int dy int dz) in Point3D.

Problemi con l'ereditarietà



Modificatori: visibilità

public	visibile da tutti
(non def.)	visibile da tutti nello stesso package
protected	visibile dalle sottoclassi
private	nascosta da tutti

Uso di metodi "di accesso":

```
public class  ACorrectClass {
    private String  aUsefulString;
    public String  getAUsefulString() {
        // "get" the value
        return aUsefulString;
    }
    private protected void setAUsefulString(String s)
{
        // "set" the value
        aUsefulString = s;
   }
}
```

Modificatori: final

- **Variabili dichiarate final sono costanti.**
- **Metodi dichiarati final non possono essere sovrascritti**
- **Classi dichiarate final non possono essere subclassate.**

Parametri di ingresso

I parametri del main sono inclusi in un vettore di String

```
/* sum and average command lines */
class SumAverage {
  public static void main (String args[]) {
    int sum = 0;
    float avg = 0;
    for (int i = 0; i < args.length; i++) {
      sum += Integer.parseInt(args[i]);
    }
    System.out.println("Sum is: " + sum);
    System.out.println("Average is: "
        + (float)sum / args.length);
  }
}
```

prompt> java SumAverage 7 9 12

Arrays

E' possibile definire arrays di tutti i tipi di dati (elementari o classi). In fase di DEFINIZIONE non e' necessario specificare la dimensione del vettore.

Solo al momento della ALLOCAZIONE viene richiesto lo spazio desiderato.

```
String[ ] strings; // this variable can refer to any String array
strings = new String[10];   // one that contains 10 Strings
strings = new String[20];   // or one that contains 20.

int f[ ][ ] = new int[5][3]; //array bidimensionale

char s[]={'+','-','*','/','=','C'}; // array inizializzato in creazione
```

Convenzioni

I nomi delle Classi iniziano con la MAIUSCOLA

I nomi degli Oggetti iniziano con la MINUSCOLA

```
Pila p=new Pila();
```

Passaggio di parametri

Le variabili dei tipi di dati primitivi sono sempre passati per copia.

Gli oggetti sono sempre passati per referenza.

(a pensarci, é ovvio: si *copia* l'identificatore dell'oggetto)

Passaggio di parametri

```
public class Numero {
   public int valore=0;
   Numero(int valore) {
    this.valore=valore;
   }
}
```

Passaggio di parametri

```
public class Parametri {
  void incrementa(int x) {x++;}
  void incrementa(Numero x) {
      x.valore++;}
  public static void main(String a[]){
    Parametri p=new Parametri();
  }
  Parametri() {
    int z=5;
    incrementa(z);
    System.out.println(z);
    Numero n=new Numero(z);
    incrementa(n);
    System.out.println(n.valore);
  }
}
```

```
public class Numero {
  public int valore=0;
  Numero(int valore) {
   this.valore=valore;
  }
}
```

Output:

5

6

Class String

java.lang

Class String

java.lang.Object
  |
  +--**java.lang.String**

All Implemented Interfaces:

CharSequence, Comparable, Serializable

---

public final class **String**
extends Object
implements Serializable, Comparable, CharSequence

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Class String

Constructor Summary
String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] ascii, int hibyte) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length) Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.
String(byte[] ascii, int hibyte, int offset, int count) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length, String charsetName) Constructs a new String by decoding the specified subarray of bytes using the specified charset.
String(byte[] bytes, String charsetName) Constructs a new String by decoding the specified array of bytes using the specified charset.
String(char[] value) Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count) Allocates a new String that contains characters from a subarray of the character array argument.
String(String original) Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Class String

Method Summary	
char	**charAt**(int index) Returns the character at the specified index.
int	**compareTo**(Object o) Compares this String to another Object.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contentEquals**(StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(String anotherString) Compares this String to another String, ignoring case considerations.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the the getBytes() method, which uses the platform's default charset.*

Class String

Constructor Summary
String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] ascii, int hibyte) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length) Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.
String(byte[] ascii, int hibyte, int offset, int count) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length, String charsetName) Constructs a new String by decoding the specified subarray of bytes using the specified charset.
String(byte[] bytes, String charsetName) Constructs a new String by decoding the specified array of bytes using the specified charset.
String(char[] value) Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count) Allocates a new String that contains characters from a subarray of the character array argument.
String(String original) Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Class String

Method Detail

length

```
public int length()
```

Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Specified by:

`length` in interface `CharSequence`

Returns:

the length of the sequence of characters represented by this object.

---

charAt

```
public char charAt(int index)
```

Returns the character at the specified index. An index ranges from `0` to `length() - 1`. The first character of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the character.

Returns:

the character at the specified index of this string. The first character is at index `0`.

Throws:

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

String

- Per trasformare il contenuto di una stringa in un intero:
- int Integer.parseInt(String s)

- Per trasformare il contenuto di una stringa in un float:
- float Float.parseFloat(String s)

Javadoc

Input

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The
name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url an absolute URL giving the base location of the image
 * @param name the location of the image, relative to the url
argument
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name) {
   try { return getImage(new URL(url, name));
   } catch (MalformedURLException e) { return null; }
}
```

Output

getImage

public Image **getImage**(URL url, String name)

Returns an Image object that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument. This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url - an absolute URL giving the base location of the image

name - the location of the image, relative to the url argument

Returns:

the image at the specified URL

See Also:

Image

Tags

Include tags in the following order:

@author (classes and interfaces only, required)
@version (classes and interfaces only, required.)
@param (methods and constructors only)
@return (methods only)
@exception (@throws is a synonym added in Javadoc 1.2)
@see (additional references)
@since (since what version/ since when is it available?)
@serial (or @serialField or @serialData)
@deprecated (why is deprecated, since when, what to use)

Documentation generation

To generate the html documentation, run javadoc followed by the list of source files, which the documentation is to be generated for, in the command prompt (i.e. *javadoc [files]*).

javadoc also provides additional options which can be entered as switches following the javadoc command (i.e. *javadoc [options] [files]*).

javadoc options

Here are some basic javadoc options:

- ***-author*** **- generated documentation will include a author section**
- ***-classpath [path]*** **- specifies path to search for referenced .class files.**
- ***-classpathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for referenced .class files.**
- ***-d [path]*** **- specifies where generated documentation will be saved.**
- ***-private*** **- generated documentation will include private fields and methods (only public and protected ones are included by default).**
- ***-sourcepath [path]*** **- specifies path to search for .java files to generate documentation form.**
- ***-sourcepathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for .java files to generate documentation form.**
- ***-version*** **- generated documentation will include a version section**

Examples

Basic example that generates and saves documentation to the current directory (c:\MyWork) from A.java and B.java in current directory and all .java files in c:\OtherWork\.

c:\MyWork> javadoc A.java B.java c:\OtherWork*.java

More complex example with the generated documentation showing version information and private members from all .java files in c:\MySource\ and c:\YourSource\ which references files in c:\MyLib and saves it to c:\MyDoc.

c:\> javadoc -version -private -d c:\MyDoc
-sourcepathlist c:\MySource;c:\YourSource
-classpath c:\MyLib

More info

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

The javadoc tool does not directly document anonymous classes -- that is, their declarations and doc comments are ignored. If you want to document an anonymous class, the proper way to do so is in a doc comment of its outer class