Sezione: Upcast - downcast

Upcast & downcast

cast

```
public class Test {
   public static void main(String a[]) {
      new Test();
   }
   Test() {
      A a;
      B b = new B();
      a=b;
       a.f1();
       a.f2();
  }
}
```

OK: upcast implicito

NO: "method f2 not found in class A" (compiler)

```
class A            { void f1()
  {System.out.println("f1");} }
class B extends A { void f2()
  {System.out.println("f2");} }
class C extends B { void f3()
  {System.out.println("f3");} }
```

cast

```
public class Test {
   public static void main(String a[]) {
      new Test();
   }
   Test() {
      A a;
      B b = new B();
      a=b;
       a.f1();
       ((B)a).f2();
   }
}
```

OK: upcast implicito

OK: downcast corretto

```
class A             { void f1()
   {System.out.println("f1");} }
class B extends A { void f2()
   {System.out.println("f2");} }
class C extends B { void f3()
   {System.out.println("f3");} }
```

cast

```
public class Test {
   public static void main(String a[]) {
      new Test();
   }
   Test() {
      A a;
      B b = new B();
      a=b;
       a.f1();
       ((C)a).f3();
   }
}
```

OK: upcast implicito

NO: downcast illecito (runtime)
java.lang.ClassCastException

```
class A             { void f1()
   {System.out.println("f1");} }
class B extends A { void f2()
   {System.out.println("f2");} }
class C extends B { void f3()
   {System.out.println("f3");} }
```

Type conversion - cast

Si può applicare cast SOLO all'interno di una gerarchia di ereditarietà

È consigliabile usare l'operatore instanceof per verificare prima effettuare un downcast

```
if (staff[1] instanceof Manager) {
      Manager n = (Manager)staff[1];
      ...
}
```

La Pila in Java – 8a

```
public static void main(String args[]) {
  int dim=10;
  Pila s=new Pila();
  //INSERIMENTO
  for (int k=0;k<dim;k++){
      Object o;
      if (Math.random()<0.5)
          o=new Integer(k);
       else
          o=new Float(k*Math.PI);
      s.inserisci(o);
  }
```

La Pila in Java – 8b

```
  // ESTRAZIONE
  for (int k=0;k<dim;k++) {
    Object o = s.estrai();
    if (o instanceof Integer) {
      Integer i = (Integer) o;
      int w = i.intValue();
      System.out.println("an int:"+w);
    } else if (o instanceof Float) {
      Float i = (Float) o;
      float w = i.floatValue();
      System.out.println("a float:"+w);
    } else
      System.out.println("Unknown class!");
  }
}
```

La Pila in Java – 8c

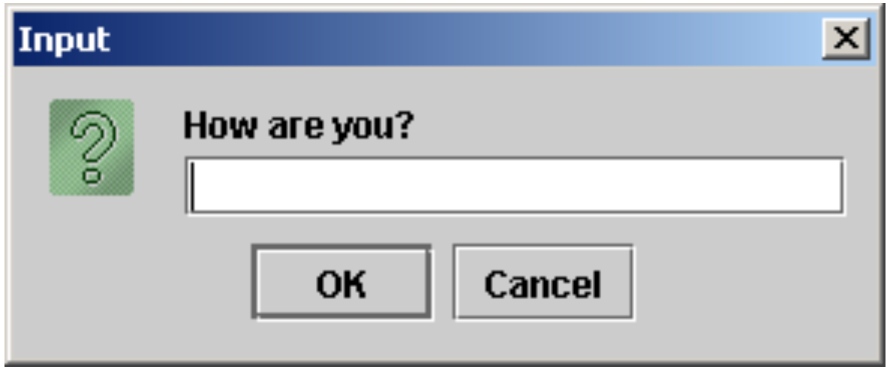
```
OUTPUT:

a float:28.274334
an int:8
an int:7
a float:18.849556
an int:5
an int:4
a float:9.424778
a float:6.2831855
a float:3.1415927
a float:0.0
```

Lettura di stringhe con GUI

```
import javax.swing.JOptionPane;
public A()  {
    ...
    String input = JOptionPane.showInputDialog(
      "How are you?");
    System.out.println(input);
    System.exit(1);
}
```

Essenziale!
Altrimenti la thread che gestisce la GUI rimane viva, e il processo non termina



Fondamenti di Java

Polimorfismo a tutto campo,
con Pile e Code...

Trasformare la Pila in Coda

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

Usare Pile e Code

```
public static void main(String args[]) {
   try {
      Pila s=null;
      int type=0;
      do {
         try {
            type =Integer.parseInt(
                JOptionPane.showInputDialog(
                "Pila (1) o Coda (2)?"));
          } catch (Exception e) {type=0;}
        } while (type<1 || type>2);
        switch (type) {
          case 1: s=new Pila(); break;
          case 2: s=new Coda(); break;
        }
```

Polimorfismo!

Usare Pile e Code

```
      for (int k=0;k<10;k++)
         if (k%2!=0)
            s.inserisci(new Integer(k));
         else
            s.inserisci(new Float(k*Math.PI));
      for (int k=0;k<11+1;k++)
         System.out.println(s.estrai());
   } catch (AssertionError a){
      a.printStackTrace();
   } finally {
      System.exit(0);
   }
}
```

Polimorfismo!

Dynamic binding

Coercion

Una funzione può essere polimorfa senza essere stata disegnata tale intenzionalmente.

Sia *f* una funzione che prende un argomento di tipo *T*, e *S* sia un tipo che può essere *automaticamente convertito* in *T*. Allora *f* può essere detta polimorfa respetto a *S* e *T*.

float somma(float x, float y)
accetta anche
somma (3, 3.14)
somma(2,3)
(coercion di int a float)

Modificatori: visibilità

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

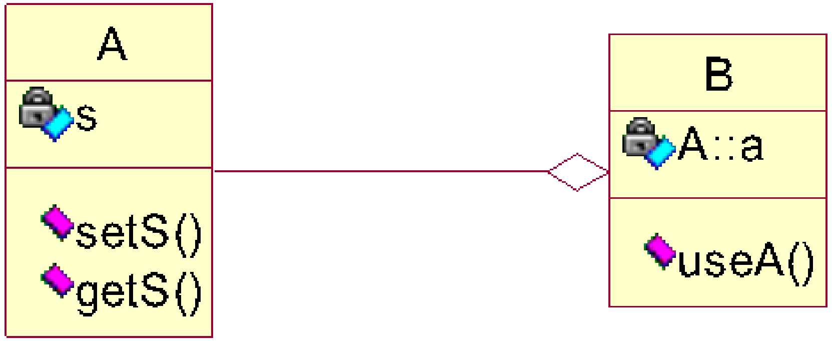
Uso di metodi “di accesso”:

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

   }
}
```

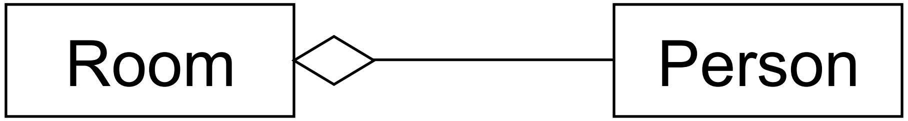
UML: Aggregazione

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

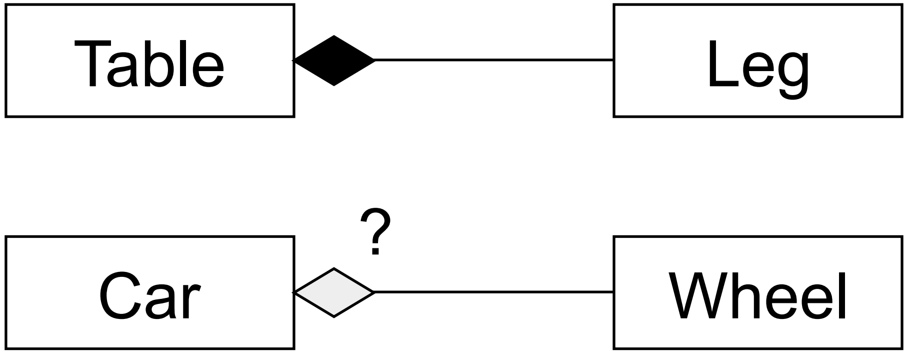


Aggregation - Composition

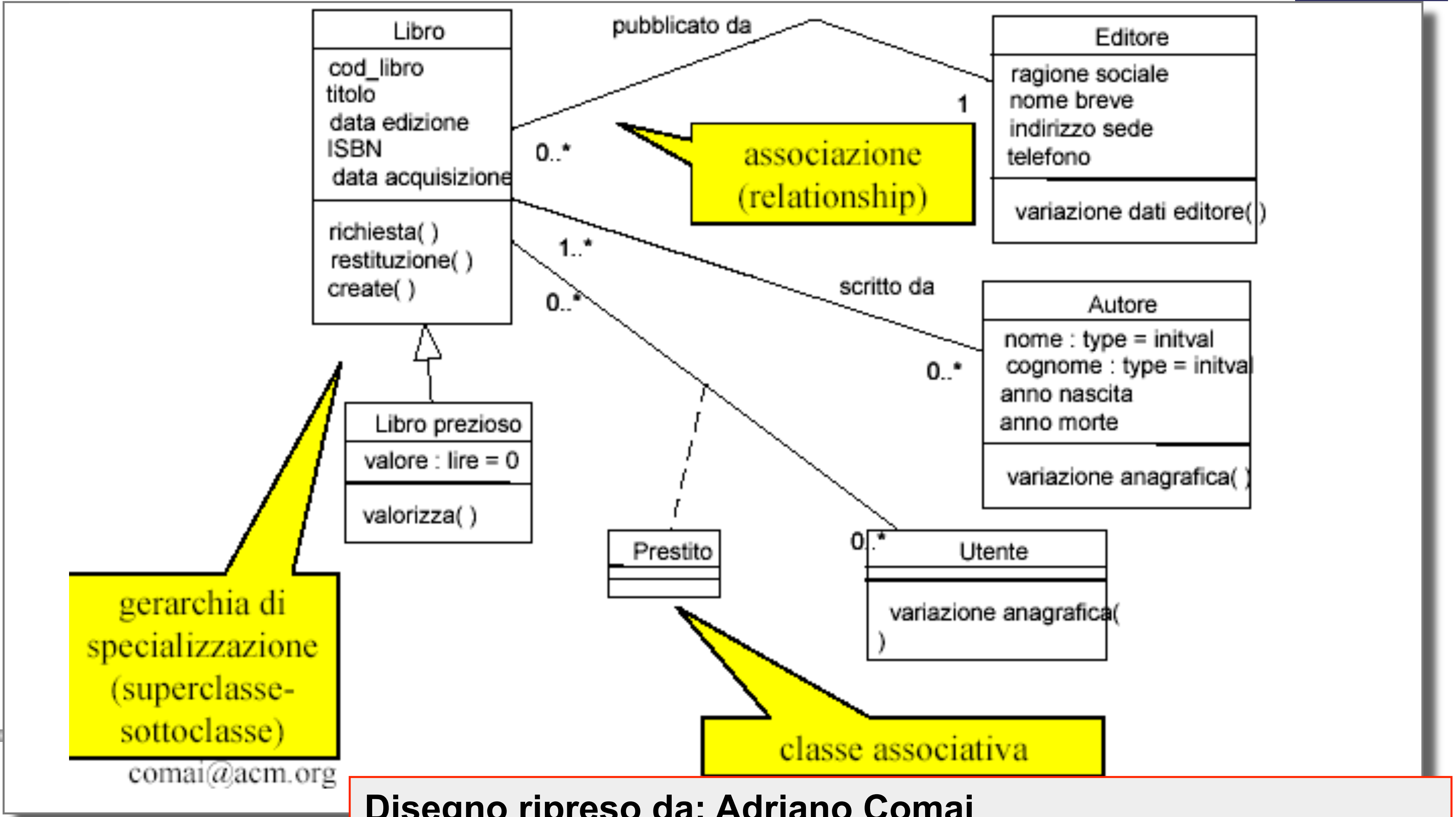
Use *aggregation (has-a)* when the lifecycle of the partecipating elements is different (one can exist without the other).



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

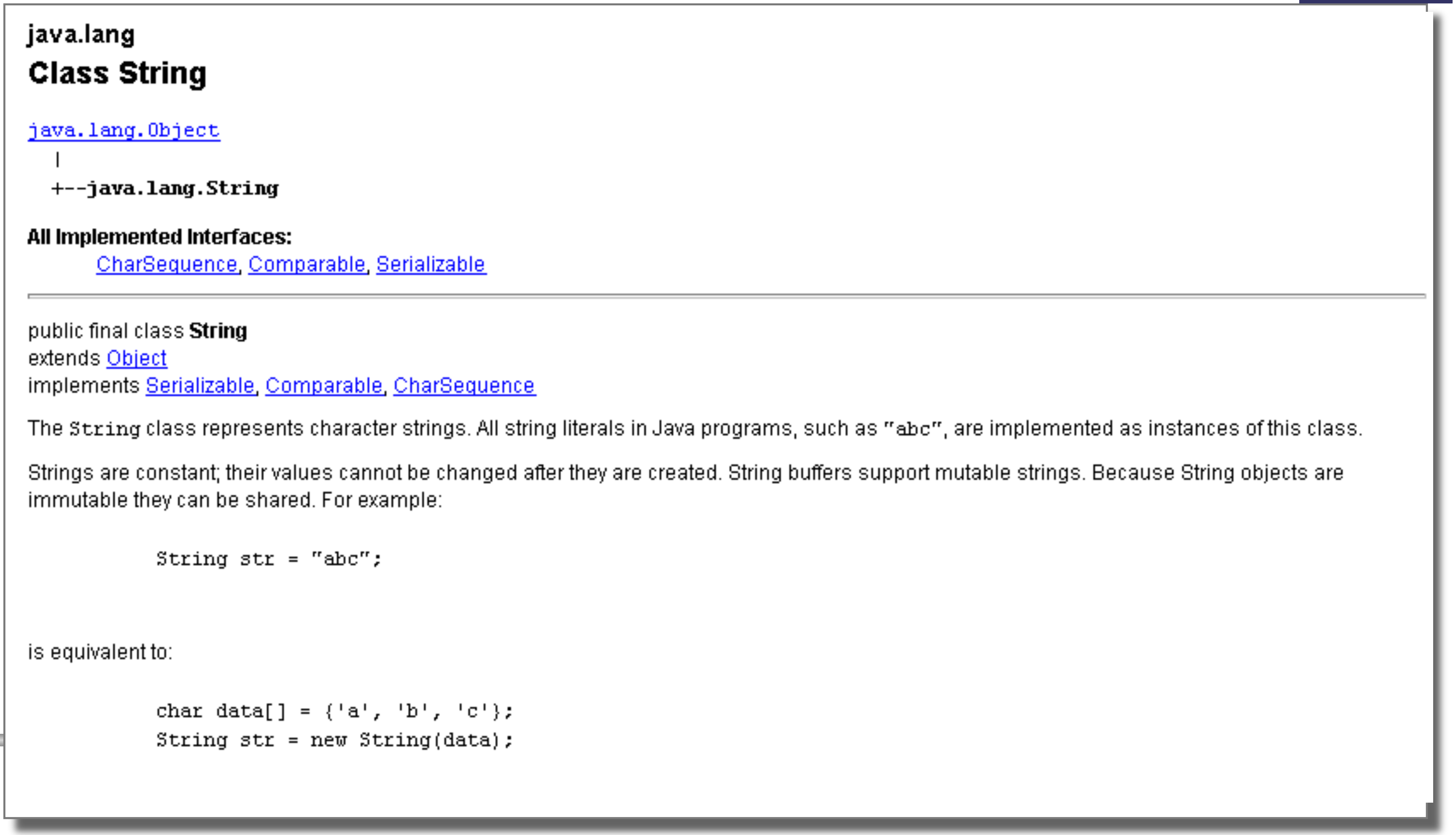


UML – Class Diagram



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

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.

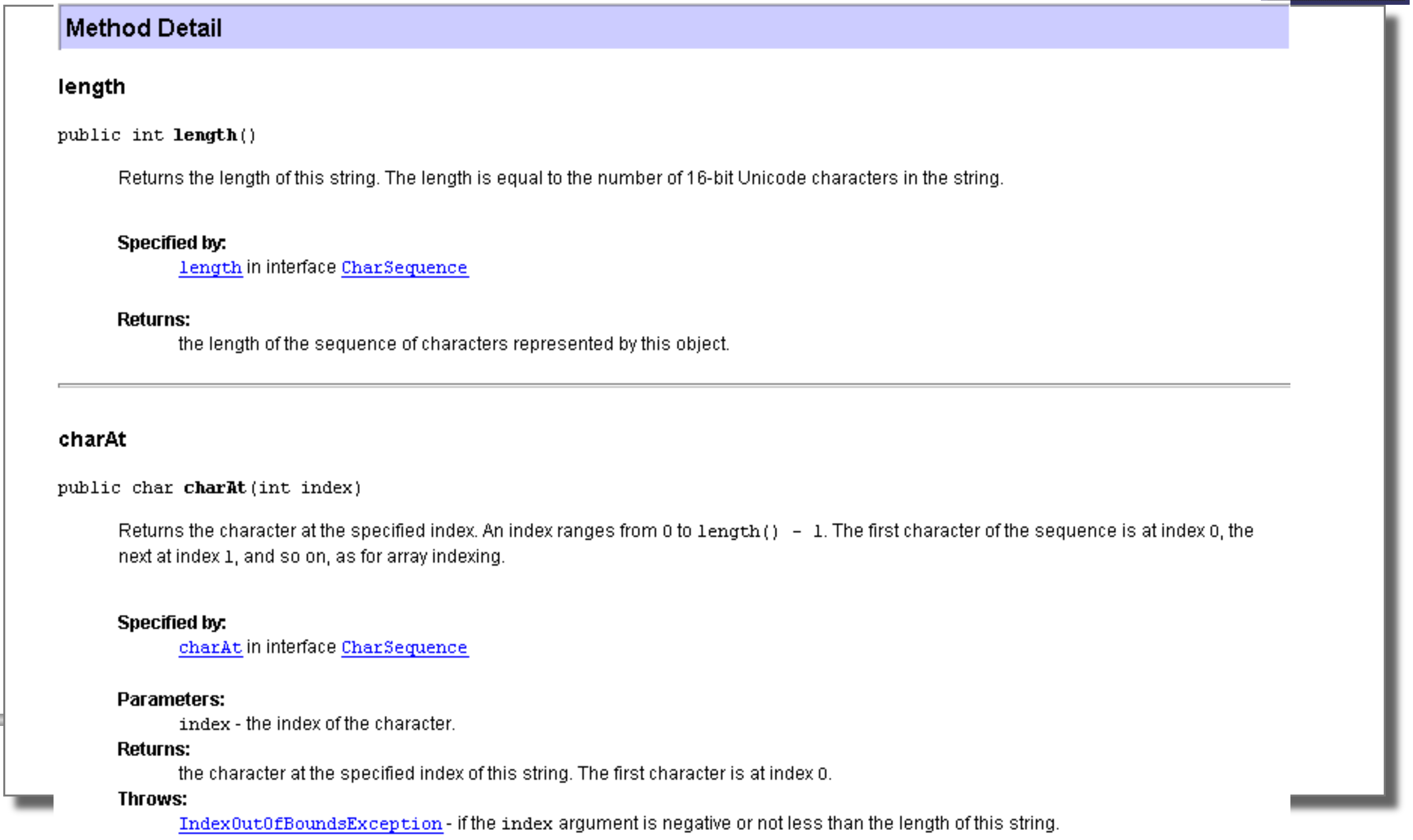
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

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

is equivalent to:

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

Class String



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)

Lettura di stringhe

```
import java.io.*;
public class A {
  public A()  {
    BufferedReader s = new BufferedReader(
            new InputStreamReader(System.in));
    try {
       System.out.println("Dammi una stringa");
       String str=s.readLine();
       System.out.println("Hai scritto "+str);
    }catch (IOException e) {e.printStackTrace();
    }
  }
  public static void main(String [] ar) {
    A a=new A();
  }
}
```

```
Dammi una stringa
abracadabra
Hai scritto abracadabra
```

Lettura di int

```
public A()  {
    int i=0;
    BufferedReader s = new BufferedReader(
           new InputStreamReader(System.in));
    try {
       System.out.println("Dammi un intero");
       i=Integer.parseInt(s.readLine());
       System.out.println("Hai scritto "+i);
    }catch (Exception e) {e.printStackTrace();}
}
```

```
Dammi un intero
2
Hai scritto 2
```

```
public A()  {
    int i=0;
    BufferedReader s = new BufferedReader(
           new InputStreamReader(System.in));
    try {
       System.out.println("Dammi un intero");
       i=Integer.parseInt(s.readLine());
       System.out.println("Hai scritto "+i);
    }catch (IOException e) {e.printStackTrace();}
}
```

```
Dammi un intero
pippo
java.lang.NumberFormatException: For input string: "gh"
  at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
  at java.lang.Integer.parseInt(Integer.java:426)
  at java.lang.Integer.valueOf(Integer.java:532)
  at pila.A.<init>(A.java:11)
  at pila.A.main(A.java:19)
Exception in thread "main"
```

```
Dammi un float
pippo
Input non valido
Dammi un float
3
Hai scritto 3.0
```

```
public A()  {
    float f=0; boolean error;
    BufferedReader s = new BufferedReader(
           new InputStreamReader(System.in));
    try {
        do {
           System.out.println("Dammi un float");
           try{
              error=false;
              f=Float.parseFloat(s.readLine());
           } catch (NumberFormatException e) {
              error=true;
              System.out.println("Input non valido");
           }
         } while (error);
         System.out.println("Hai scritto "+f);
    }catch (IOException e) {e.printStackTrace();}
}
```

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);
  }
}
```