JAVA
IL LINGUAGGIO

Tipi e variabili

- Ogni valore nel linguaggio ha un tipo
- Ogni variabile deve essere dichiarata ed associata ad un tipo:

```
String greeting = "Hello, World!";
PrintStream printer = System.out;
int luckyNumber = 13;
```

- Il tipo determina quali sono le operazioni legali sui valori (e le variabili)
- Variabili di un tipo possono solo assumere valori di quel tipo

Controllo dei tipi forte

- **Il compilatore controlla che il programma rispetti strettamente tutte le regole**
  - controlla l'uso dei cast
- **I programmi che superano i controlli del compilatore non causano errori nelle operazioni sui dati**

Oggetti e classi

OGGETTI

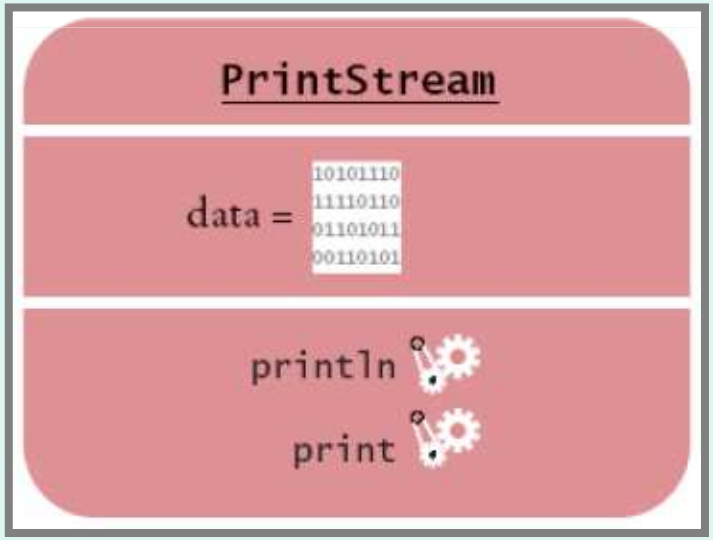
- **Le strutture dati complesse di un programma**
  - in C: structs, unions e arrays
  - In Java: oggetti (arrays inclusi)
- **Generalizzazione delle struct di C**
  - campi: struttura dell'ggetto
  - metodi: operazioni sui campi

CLASSI

- Descrivono la struttura dei campi
- Definiscono l'implementazione dei metodi
- Sono i tipi degli oggetti

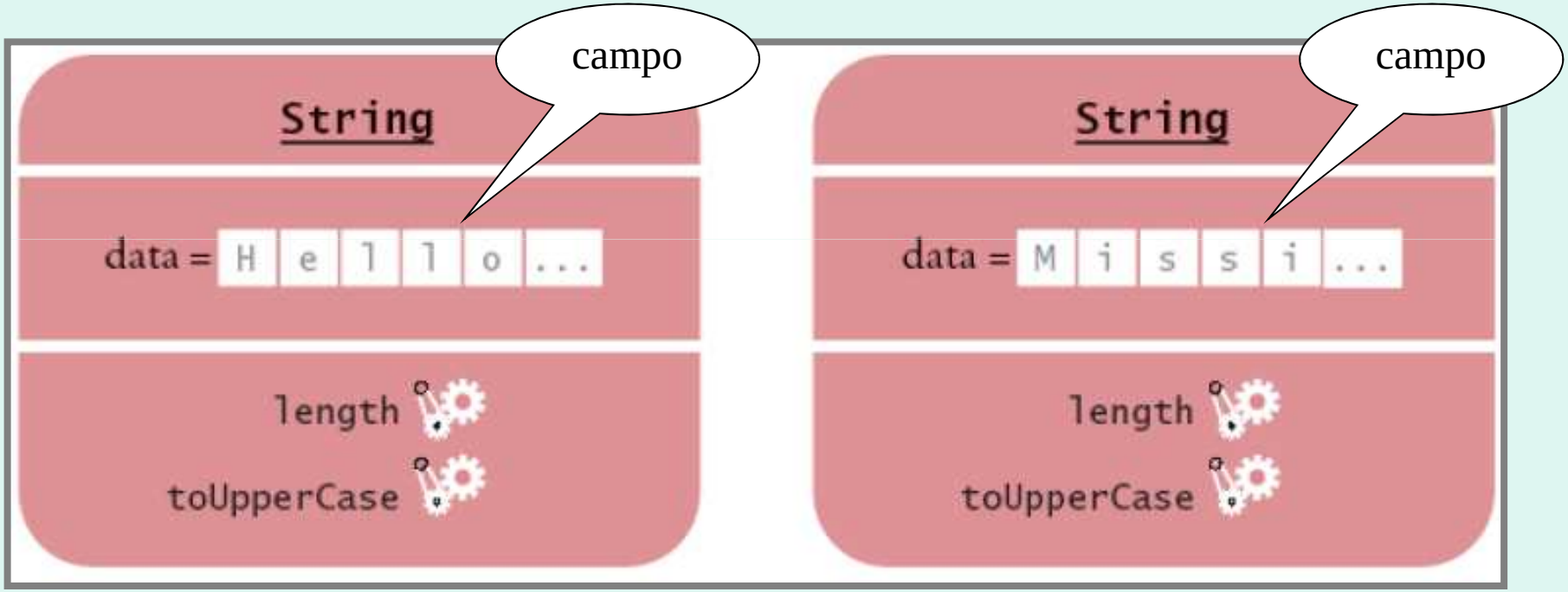
Oggetti e classi

- **Ogni oggetto appartiene ad una classe.**
- **un oggetto di classe `PrintStream`**



Due oggetti di tipo `String`

- **Ogni oggetto ha una copia privata dei suoi campi**

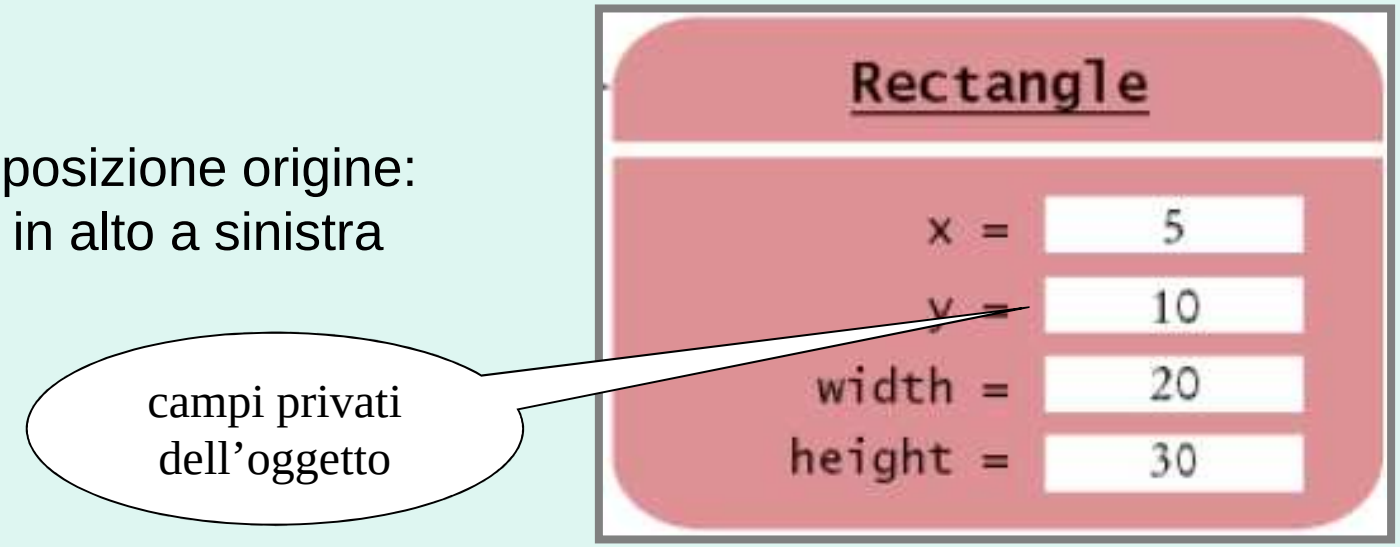


- **Tutti gli oggetti di una classe condividono lo stesso codice dei metodi**

Oggetti di classe `Rectangle`

- **Classe definita nella libreria `java.awt`**
- **Descrive la struttura di un rettangolo posizionato sul piano cartesiano**

(x,y) = posizione origine:
angolo in alto a sinistra



Costruttori

```
new Rectangle(5, 10, 20, 30)
```

- **Il costrutto `new` invoca il costruttore, per allocare nuovo oggetto di tipo `Rectangle`**
  - Il nome del costruttore coincide con il nome della classe
  - Usa i parametri (`5`, `10`, `20`, e `30`) per inizializzare i dati dell'oggetto
- **Restituisce un riferimento all'oggetto**

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Costruttori

- **Una classe può fornire più di un costruttore**
- ***Overloading***
- **Diverse modalità di creazione di oggetti**

```
Rectangle box1 = new Rectangle(5, 10, 20, 30)
```

```
Rectangle box2 = new Rectangle()
    // costruisce un rettangolo con origine (0,0)
    // larghezza 0, e altezza zero 0
```

Sintassi: new

```
new ClassName(parameters)

Esempi:
new Rectangle(5, 10, 20, 30)
new Rectangle()
```

- **Costruisce un nuovo oggetto, inizializza lo stato con i parametri, e restituisce un riferimento all'oggetto costruito.**

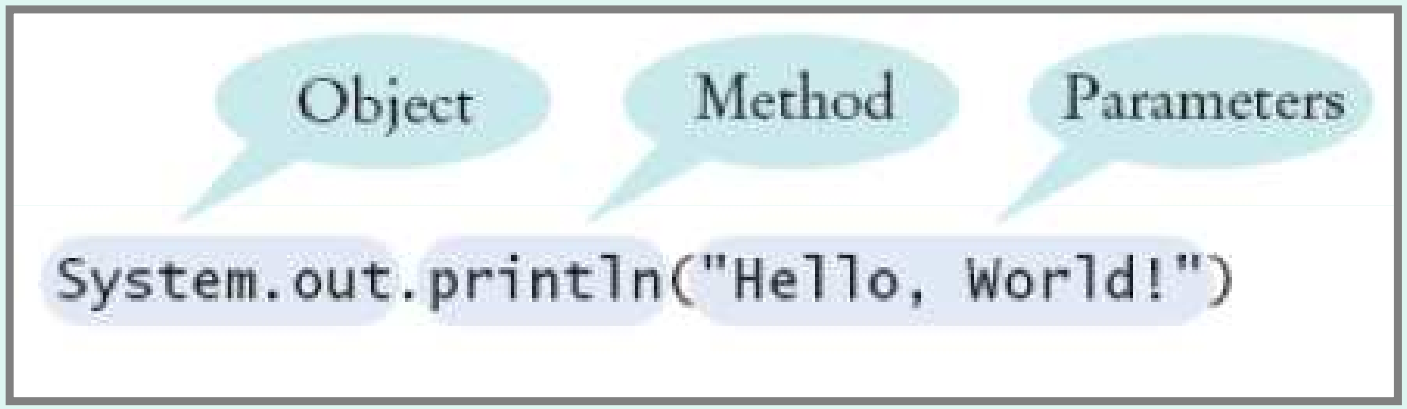
Metodi

- **Realizzano le operazioni supportate dagli oggetti**
- **Definiti dalla classe**
  - Classe definisce l'implementazione
  - Descrive la firma visibile all'esterno
- **Invocazione**
  - Deve indicare l'oggetto "target"

```
oggetto.metodo(argomenti)
```

a...

Metodi della classe `PrintStream`



Metodi della classe `String`

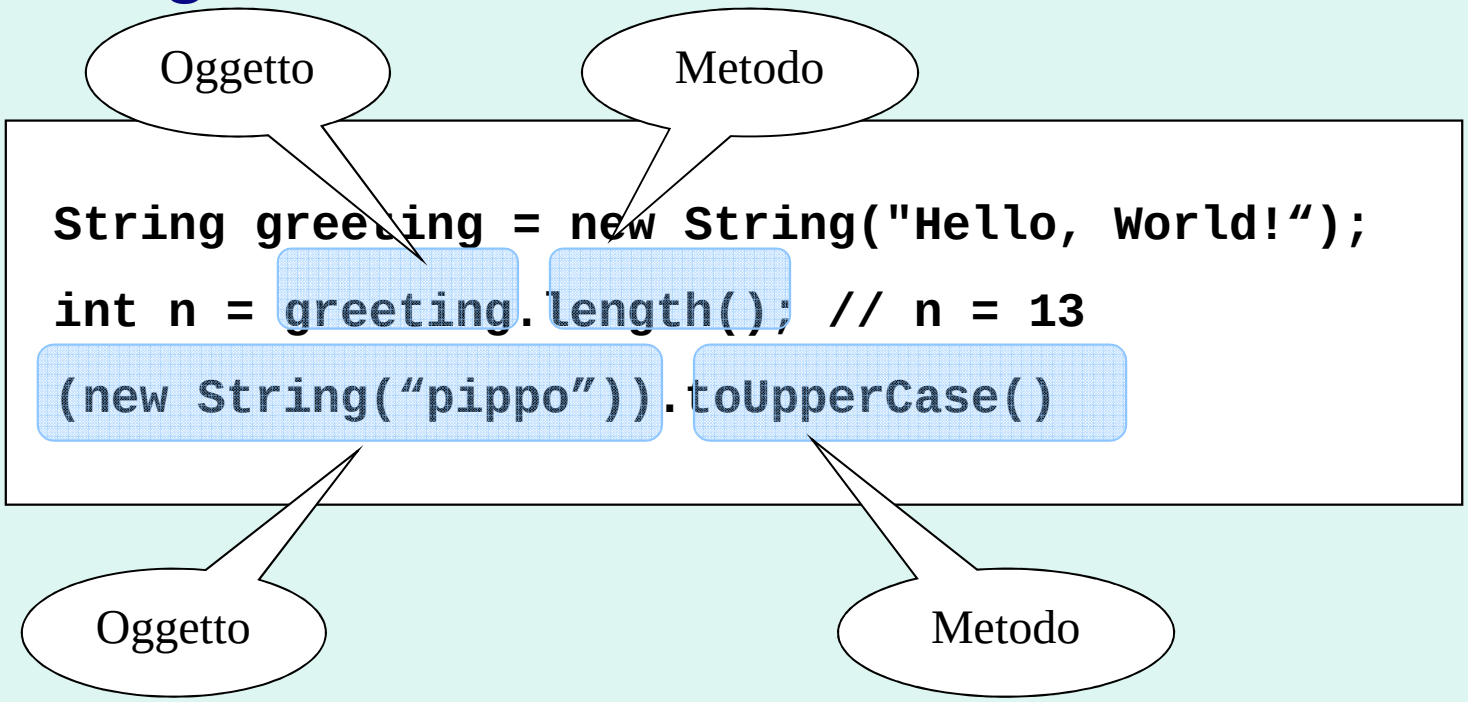
- **`toUpperCase`: crea un altro oggetto di tipo `String` che contiene i caratteri della stringa originaria, ma convertiti in maiuscolo**

```
String fiume = "Mississippi";
String fiumeInUpperCase = fiume.toUpperCase();
// fiumeInUpperCase = "MISSISSIPPI"
```

Continua…

Metodi della classe `String`

- **`length()`: conta il numero di caratteri della stringa**



Domande

- Quale è la sequenza di istruzioni per calcolare la lunghezza della stringa “`Arcobaleno`”?
- Quale è la sequenza di istruzioni per stampare la versione uppercase della stringa `"Hello, World!"`?
- È legale la seguente sequenza di istruzioni?

```
String fiume = "Mississippi";
fiume.println();
```

Perché o perché no?

Risposte

- ```
  new String("Arcobaleno").length()
  ```
- ```
  System.out.println("Hello World".toUpperCase());
  ```
- Non è legale: la variabile `fiume` ha tipo `String` e la classe `String` non definisce il metodo `println()`

Controllo di tipi forte

- La classe di un oggetto definisce quali metodi si possono invocare sugli oggetti della classe
- Il compilatore controlla che le invocazioni rispettino la dichiarazione della classe

```
System.out.length(); // ERRORE DI TIPO (COMPILE-TIME)
```

Controllo di tipi forte

- La classe di un oggetto definisce quali metodi si possono invocare sugli oggetti della classe
- I metodi di un oggetto sono l'unico modo per agire sui campi dell'oggetto

```
Rectangle box = new Rectangle(5,10,20,20).
System.out.println(box.width); // ERRORE DI TIPO
```

Metodi della classe `Rectangle`

- **`getWidth()`: restituisce il valore che corrisponde alla base del rettangolo**

```
Rectangle box = new Rectangle(5,10,20,20).
System.out.println(box.width); // ERRORE DI TIPO
System.out.println(box.getWidth()); // OK
```

Continua…

Metodi della classe `Rectangle`

- **`Translate()`: modifica l'oggetto su cui viene invocato, traslando di x a destra, e di y verso il basso**

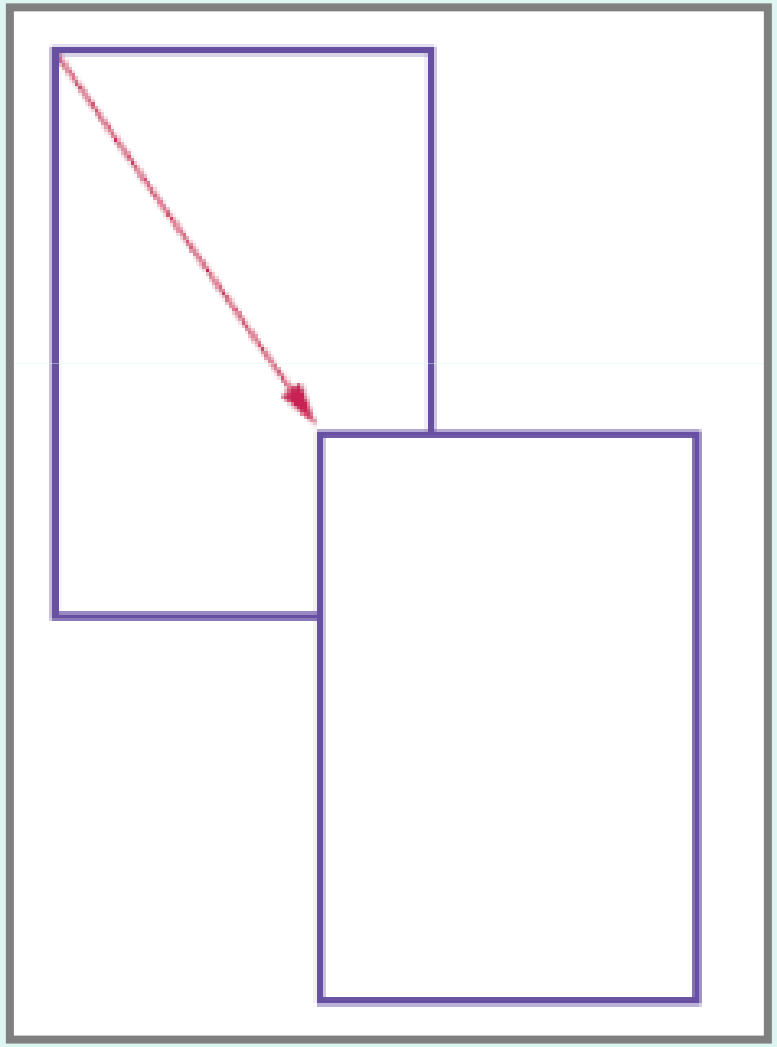
```
box.translate(x, y);
```

- ***Mutators*: metodi con side effects per modificare i valori dei campi**

Continua…

Metodi e side-effects

```
box.translate(15, 25);
```



Oggetti e riferimenti

- Un riferimento è una astrazione del puntatore ad un oggetto
- La creazione di un oggetto con **new** restituice un riferimento al nuovo oggetto

```
Rectangle box = new Rectangle();
```

- Diverse variabili di tipo oggetto possono condividere lo stesso riferimento

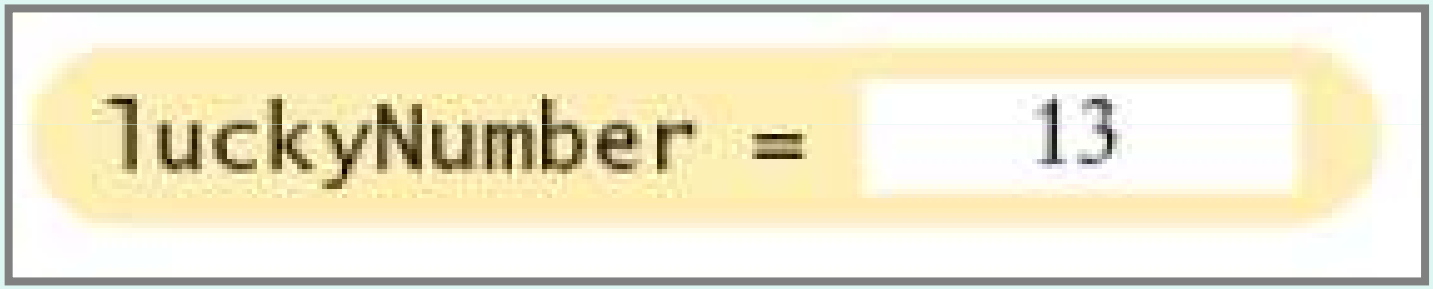
```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
```

Continua…

Assegnamento

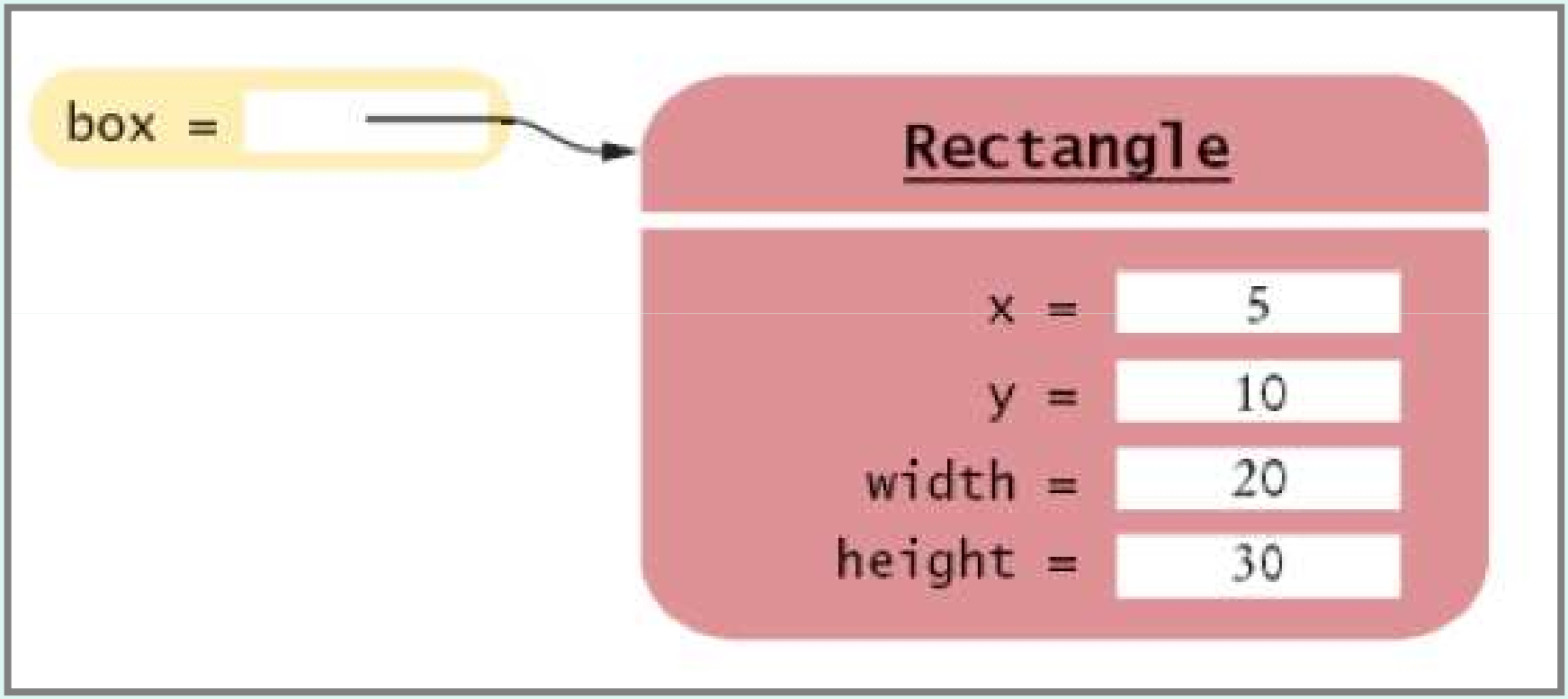
- **Il comportamento dell'operazione di assegnamento varia a seconda del tipo della variabile (e quindi del valore) coinvolto.**
- **L'assegnamento su variabili di tipo primitivo ha un effetto diverso dall'assegnamento su variabili di tipo oggetto**

Variabili di tipo primitivo



Contengono valori del loro tipo

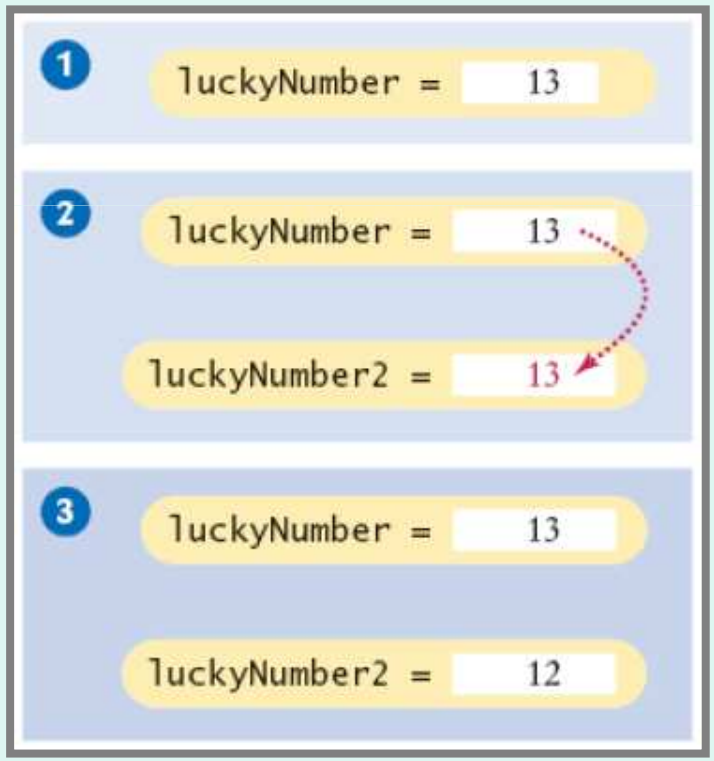
Variabili di tipo oggetto



Contengono riferimenti ad oggetti, non oggetti

Assegnamento su tipi primitivi

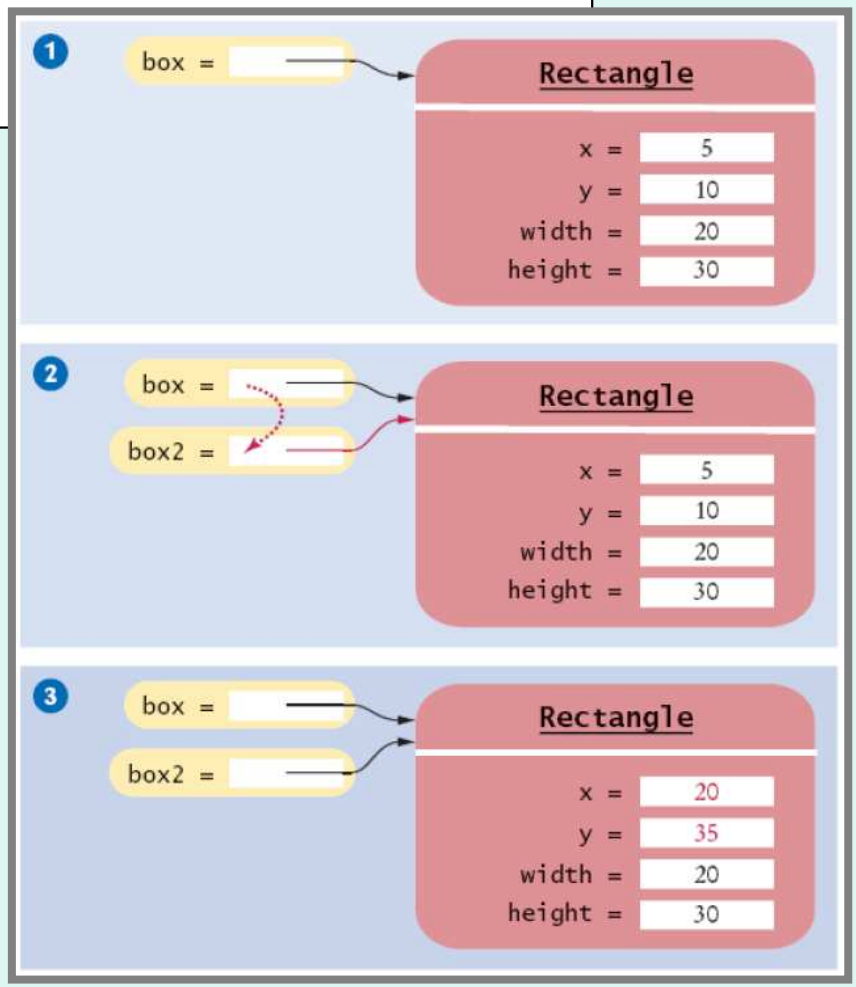
- ```
  int luckyNumber = 13;
  int luckyNumber2 = luckyNumber;
  luckyNumber2 = 12;
  ```



Due variabili, due valori distinti
  ```

Assegnamento su tipi oggetto

-
```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
```



Due variabili, stesso riferimento

Programmi e classi applicazione

Un programma è costituito da più classi

- **Classe applicazione: contiene il `main`**
  - al più una classe applicazione in un programma
- **Il `main` avvia la computazione**
  - costruisce uno o più oggetti delle del programma
  - Invoca i metodi su tali oggetti
  - Stampa i risultati

File RectangleApp.java

```
01: import java.awt.Rectangle;
02:
03: public class RectangleApp
04: {
05:    public static void main(String[] args)
06:    {
07:       Rectangle box = new Rectangle(5, 10, 20, 30);
08:
09:       // Sposta il rettangolo
10:       box.translate(15, 25);
11:
12:       // Stampa nuove info
13:       System.out.println("Origine dopo la traslazione:");
14:       System.out.println(box.getX());
15:       System.out.println(box.getY());
16:    }
17: }
```

Classi di libreria

- **Sempre includere le classi di libreria utilizzate dall'applicazione**
  - classi delle librerie raggruppate in packages
  - Importiamo le classi specificando i nomi di package e di classe

```
import java.awt.Rectangle;
```

  - Le classi del package **`java.lang`** (ad esempio **`String`** e **`System`**)sono importate automaticamente

Classi di libreria

```
import packageName.ClassName;
```

Esempio:

```
import java.awt.Rectangle; // importa la classe Rectangle
```

```
import packageName.*;
```

Esempio:

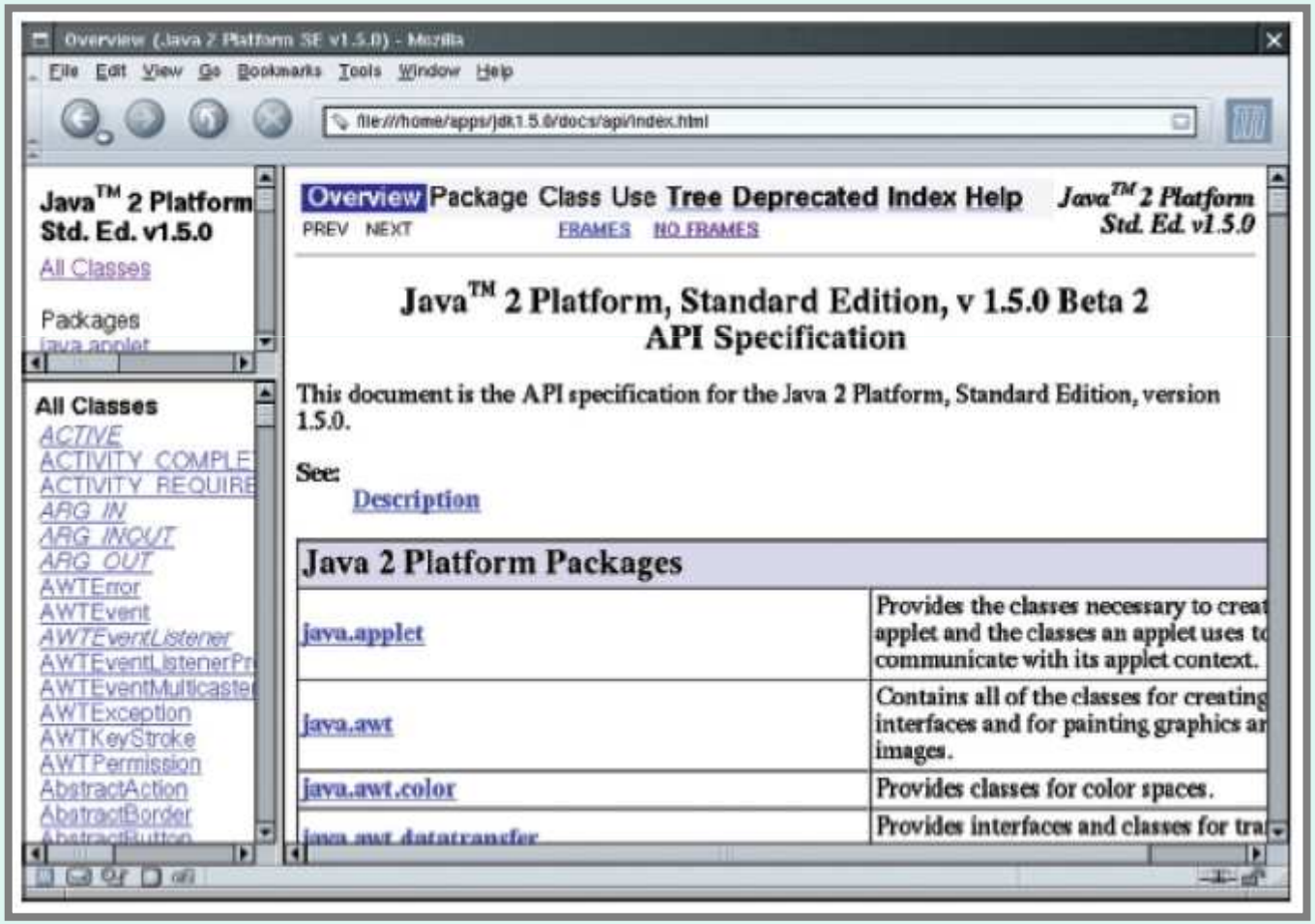
```
import java.util.*; // importa tutte le classi del package
```

Import in Java ~ #include in C

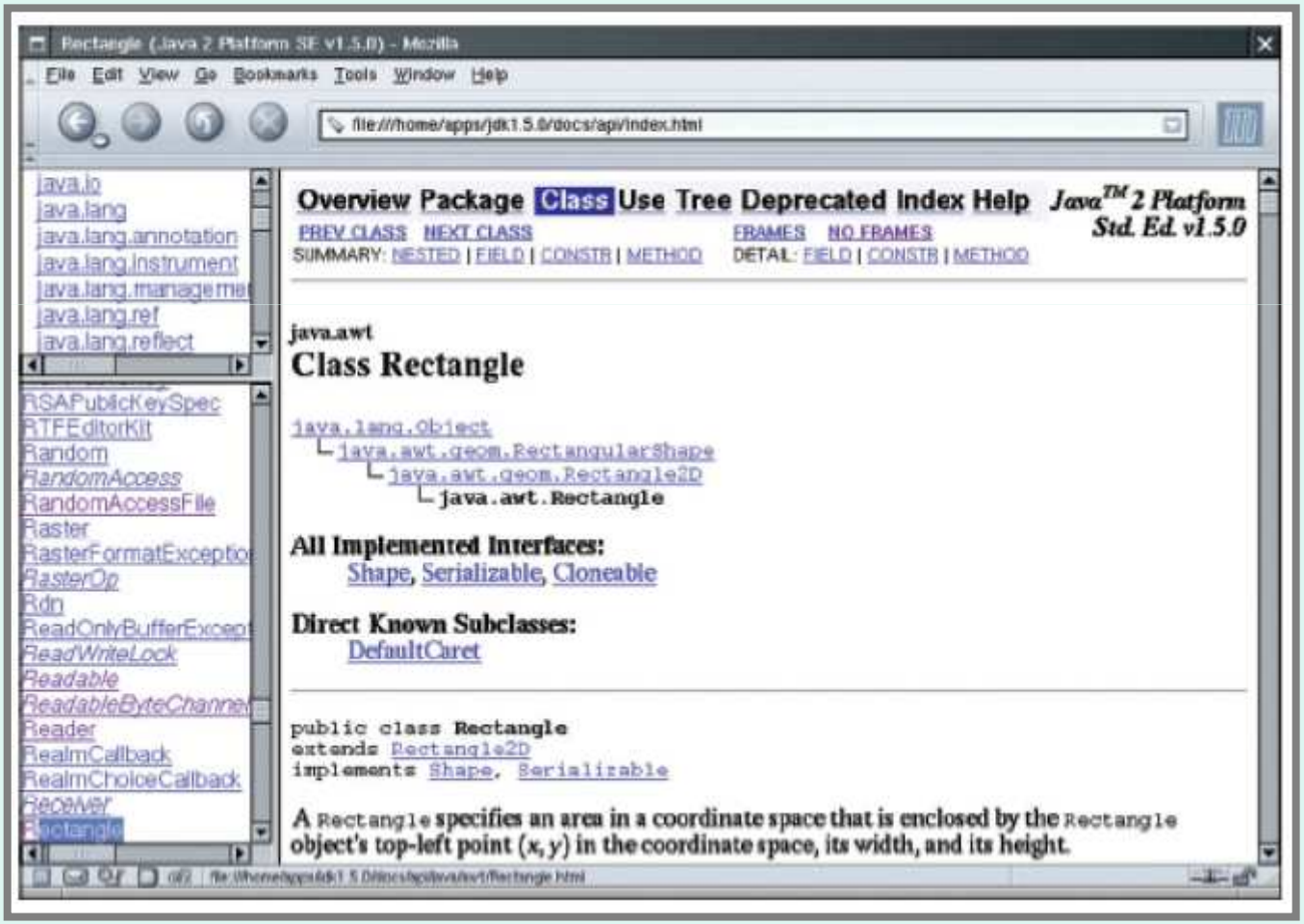
Librerie – Documentazione

- **API: Application Programming Interface**
- **Include la descrizione delle classi e dei relativi metodi della (fornitissima!) libreria Java**
- **http://java.sun.com/j2se/1.5/docs/api/index.html**
- **http://java.sun.com/j2se/1.6/docs/api/index.html**

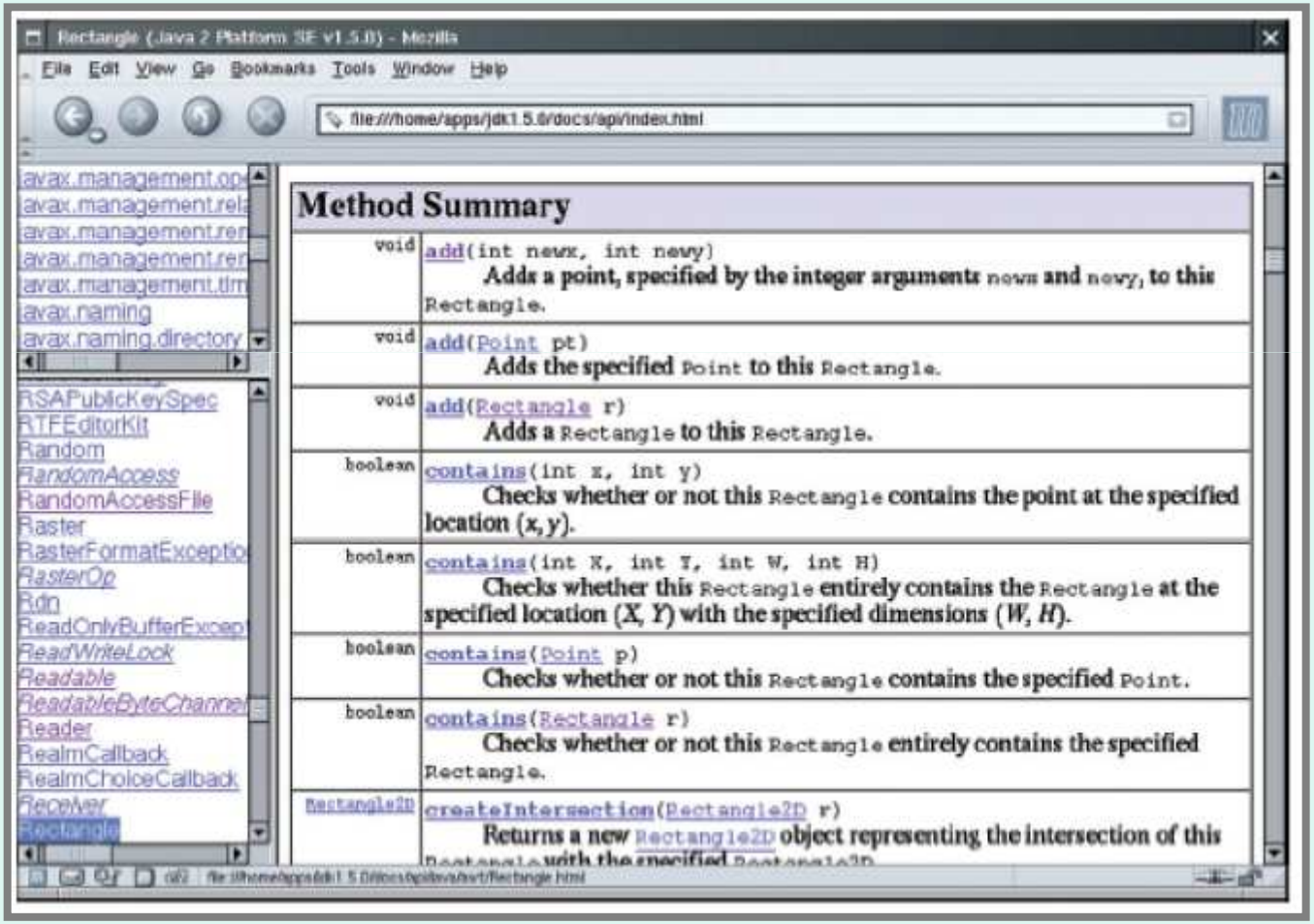
Documentazione sulle API



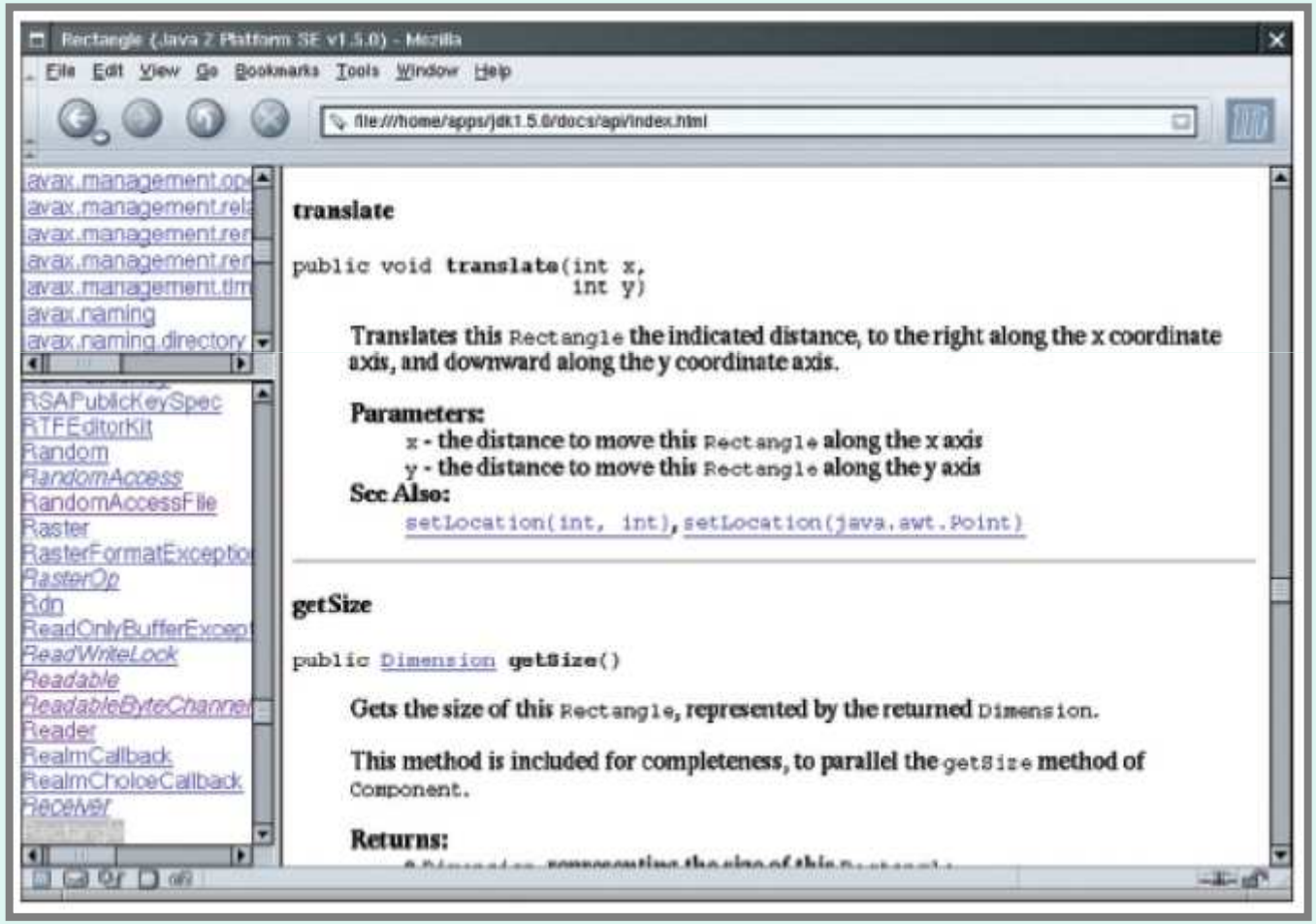
La API della classe `Rectangle`



Javadoc Method Summary



Documentazione del metodo `translate`



Domande

- **Quali sono gli errori nei seguenti frammenti di codice?**
  - `Rectangle r = (5,10,15,20)`
  - `Double w = Rectangle().getwidth();`
  - `Rectangle r; r.translate(5,10);`

Risposte

- Manca il costruttore
- Manca `new`
- `r` non è inizializzato

Esempio: Tombola!

- Vogliamo progettare una applicazione che realizza il gioco della tombola
- Versione semplificata: un banco, un giocatore, ogni giocatore una scheda
- Ci vengono già fornite le classi necessarie
  - `Banco`
  - `Giocatore`

Banco

- **Costruttore:**
  - **`Banco()`**

    crea una nuova istanza della classe
- **Metodi:**
  - **`int estraiNumero()`**

    restituisce un numero nell'intervallo [1..90]

    (ad ogni chiamata genera un numero diverso)

`Giocatore`

- **Costruttore**
  - **`Giocatore(String nome)`**

    crea un giocatore, con il nome indicato ed una sola scheda
- **Metodi:**
  - **`void controllaNumero(int x)`**

    controlla se il numero è contenuto nella sua scheda; se si lo “marca”

Giocatore

- **Metodi:**
  - **`boolean tombola()`**

    restituisce true quando tutti i numeri della sua scheda sono “marcati”
  - **`Integer[] verifica()`**

    restituisce un array che contiene i numeri della sua scheda

Integer è un tipo reference isomorfo al tipo int

Tombola

- **La classe applicazione che realizza il gioco:**
  - crea un banco ed un giocatore
  - esegue un ciclo in cui ad ogni iterazione in banco estrae un numero, il giocatore lo controlla e controlla se ha completato la sua scheda
  - il ciclo termina non appena il giocatore completa la sua scheda
  - All'uscita dal ciclo stampa i numeri della scheda del giocatore

Tombola

```
public class Tombola
{
   public static void main(String[] args)
   {
      Banco b = new Banco();
      Giocatore g = new Giocatore(“Mario”);

       while (true) {
          int x = b.estraiNumero();
          System.out.println(“Numero estratto = ” + x);
          g.controllaNumero(x);
          if (g.tombola()) break;
       }

       Integer[] scheda = g.verifica();
       for (int i = 0; i < scheda.length; i++)
          System.out.println(scheda[i])
   }
}
```

NOTE

- **Nessuna informazione data sulla struttura interna degli oggetti in gioco**
- **Conosciamo i metodi che essi forniscono**
  - notare bene: conosciamo la specifica dei metodi, non come sono implementati
- **E' tutto quello che serve per programmare l'applicazione**
- **La chiave del concetto di *encapsulation***

Oggetti / Encapsulation

- **Oggetti**
  - esportano la specifica dei propri metodi
    - interfaccia pubblica che definisce l'interazione tra oggetti e codice "cliente"
  - mascherano la propria struttura interna (i campi) e l'implementazione dei metodi
- **Codice cliente**
  - indipendente dall'implementazione degli oggetti

Esercizio

- **Create una versione della tombola in cui possono giocare più giocatori**
  - Il gioco termina non appena uno dei giocatori completa la sua scheda