

# Utilizzare gli Oggetti

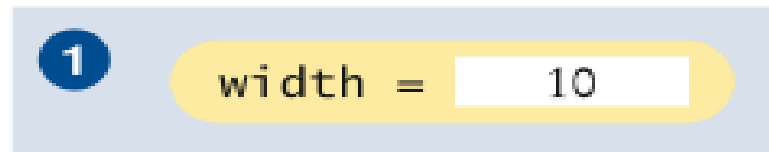
# Tipi

- I programmi manipolano dati (valori)
- Ogni dato ha un tipo
- In Java, tipi raggruppati in due categorie:
  - tipi primitivi (int, double, etc)
  - oggetti

```
"Hello, World!" oggetto di tipo String  
System.out  oggetto di tipo PrintStream  
13  valore di tipo int  
true valore di tipo boolean
```

# Tipi e variabili

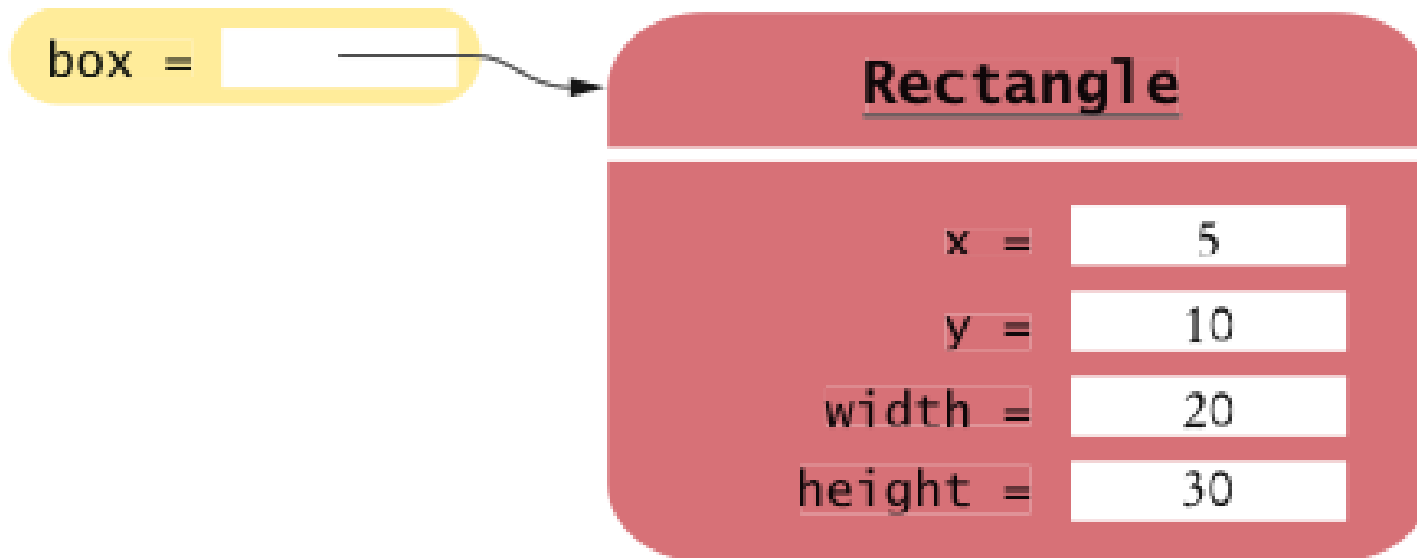
- I dati (valori) di un programma sono memorizzati attraverso le variabili
- Ogni variabile corrisponde ad una porzione di memoria
  - ❑ Per i tipi primitivi, la memoria corrispondente contiene il dato
  - ❑ Ad es. la variabile **width** di tipo int:



A diagram illustrating a variable assignment. It features a light blue rectangular background. On the left, there is a dark blue circle containing the number '1'. To the right of this circle is a yellow rounded rectangle. Inside the yellow rectangle, the text 'width =' is displayed in a monospaced font. To the right of the equals sign is a white rectangular box containing the number '10'.

# Tipi e variabili

- ❑ Per gli oggetti, la memoria corrispondente contiene l'indirizzo (riferimento) al quale è memorizzato l'oggetto



**Figure 17** An Object Variable Containing an Object Reference

# Tipi e variabili

- Per utilizzare una variabile in un programma occorre dichiararla

□ Es.

```
String greeting;  
PrintStream printer;  
int luckyNumber;  
boolean done;
```

- Per effetto della dichiarazione, ad un nome viene associata una porzione di memoria adeguata (rispetto al tipo)

# Assegnamento e valori iniziali

- Operatore di assegnamento

- `luckyNumber = 13;`

- Di solito, assegnamento valore iniziale avviene contestualmente alla dichiarazione

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

- Uso variabili non inizializzate: **errore Java!**

```
int luckyNumber;  
System.out.println(luckyNumber);  
    // ERROR - uninitialized variable
```

# Sintassi: Dichiarazione di variabili

*typeName variableName = value;*  
oppure  
*typeName variableName;*

## **Esempio:**

```
String greeting = "Hello, Dave!";
```

## **Obiettivo:**

Dichiarare una nuova variabile *variableName* di tipo *typeName* e fornire eventualmente un valore iniziale *value*

# Identificatori

- Nome di una variabile, un metodo o una classe
- Regole in Java:
  - Può contenere lettere, cifre e il carattere underscore (\_)
  - Non può cominciare con una cifra
  - Non può contenere altri simboli quali ad esempio ?, %, !, etc.
  - Gli spazi non sono consentiti
  - Non si possono usare parole riservate di Java
  - Maiuscolo/minuscolo sono significativi



# Convenzioni programmazione Java

## ■ Per convenzione:

- ❑ i nomi delle variabili cominciano per lettera minuscola
- ❑ i nomi delle classi cominciano per lettera maiuscola
- ❑ nomi composti usano maiuscola ad ogni inizio nuova parola, es:
  - contoCorrente (variabile)
  - ContoCorrente (classe)

# Oggetto

- Entità di un programma dotata di tre proprietà caratteristiche
  - ❑ stato
  - ❑ comportamento
  - ❑ identità (identifica univocamente un oggetto)
- Esempi:
  - ❑ casella vocale
  - ❑ conto corrente
  - ❑ stringa
  - ❑ studente
  - ❑ cliente

# Stato

- informazioni conservate nell'oggetto
  - Casella vocale: vuota, piena, alcuni messaggi
  - Conto corrente: saldo nullo, saldo positivo
- può condizionare il comportamento dell'oggetto nel futuro
  - Casella vocale: accetta un messaggio se e solo se non piena
  - Conto corrente: consente di prelevare se e solo se saldo positivo
- può variare nel tempo per effetto di un'operazione sull'oggetto
  - Casella vocale: aggiunta/cancellazione messaggio
  - Conto corrente: versamento/prelevamento

# Comportamento

- definito dalle operazioni (**metodi**) che possono essere eseguite dall'oggetto
  - Casella vocale: lettura messaggio, cancellazione messaggio, etc.
  - Conto corrente: lettura saldo, versamento, prelevamento, etc
- i metodi possono modificare lo stato dell'oggetto
  - Casella vocale: aggiunta messaggio può far cambiare lo stato da vuoto a alcuni messaggi, o da alcuni messaggi a pieno.
  - Conto corrente: versamento può far cambiare lo stato da saldo nullo a saldo positivo

# Oggetti immutabili

- oggetti che non modificano il loro stato
  - non hanno comportamenti che modificano lo stato
  - restano sempre nello stato che assumono nel momento dell'istanziamento
- molto utili in varie circostanze
  - semplificano l'incapsulamento dei dati e la clonazione di oggetti
  - nei programmi concorrenti non generano problemi di interferenza tra thread e consistenza della memoria
- uso di oggetti immutabili richiede un'istanziamento di un nuovo oggetto ogni volta che necessitiamo di un cambio di stato
  - istanziare oggetti non è un'operazione particolarmente onerosa

# Classe

- Ogni oggetto appartiene a (è un'istanza di) una *classe* che ne determina il tipo
- Una classe descrive un insieme di oggetti caratterizzati dagli stessi
  - possibili comportamenti (metodi)
  - possibili stati (variabili di istanza o campi)
- Es. **CasellaVocale** descrive un insieme di caselle vocali dello stesso tipo

---

# Possibili stati: le variabili di istanza

- Le variabili di istanza (campi) memorizzano lo stato di un oggetto
  - Ciascun oggetto ha la propria copia delle variabili di istanza definite nella classe di appartenenza
  - E' buona norma permettere la lettura e la modifica delle variabili di istanza soltanto attraverso i metodi della classe  
***(incapsulamento dei dati)***
-

# Possibili comportamenti: metodi

- parte computazionale della classe
- somigliano a funzioni dei linguaggi procedurali tipo C
  - possono utilizzare altri metodi (anche della stessa classe)
- possono modificare lo stato di un oggetto (contenuto delle variabili di istanza)
- se l'incapsulamento è realizzato adeguatamente si accede/modifica lo stato di un oggetto solo attraverso i suoi metodi

```
String greeting = "Hello";  
greeting.println(); // Error  
greeting.length(); // OK
```



# Alcuni metodi di String

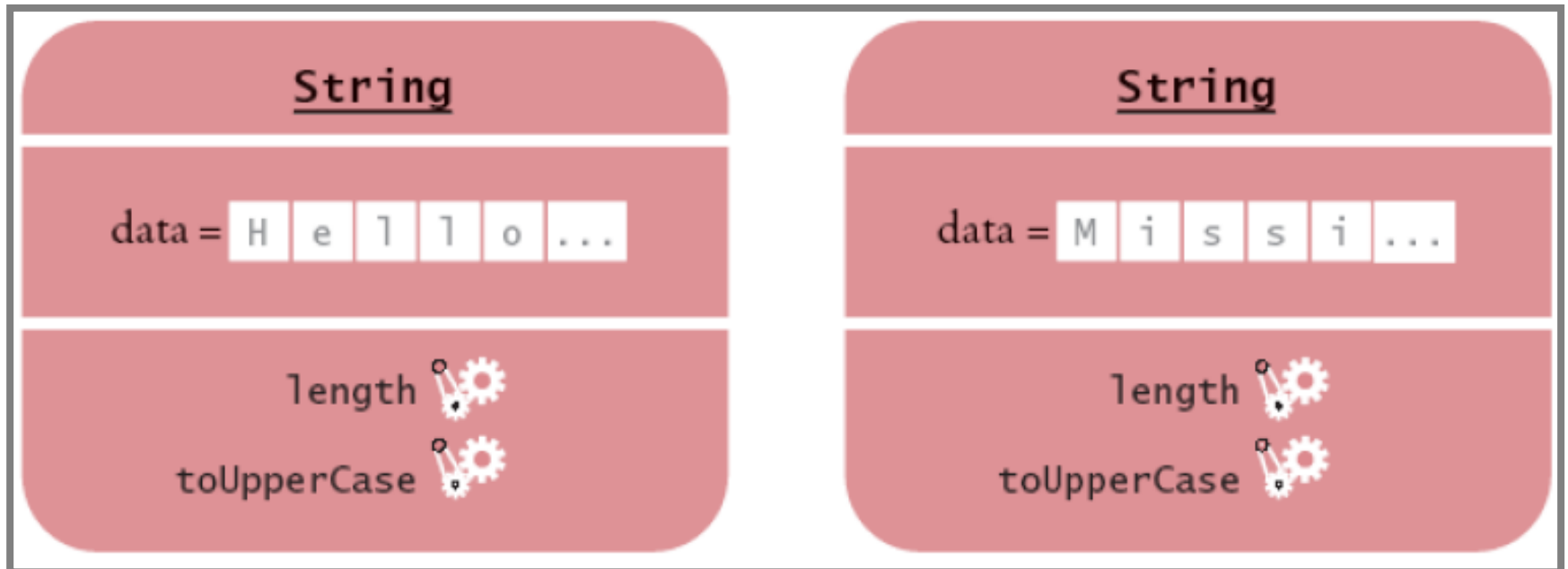
- `length()` : conta caratteri in una stringa

```
String greeting = "Hello, World!";  
int n = greeting.length(); // sets n to 13
```

- `toUpperCase()` : crea una nuova stringa che contiene i caratteri della stringa originale in maiuscolo

```
String river = "Mississippi";  
String bigRiver = river.toUpperCase();  
// sets bigRiver to "MISSISSIPPI"
```

# Rappresentazione di due oggetti String



# Invocazione di un metodo

- Per invocare un metodo su un dato **oggetto** bisogna specificare il nome del metodo preceduto dal riferimento all'oggetto e da un punto
  - Es.: `river.length()`;  
(Eseguiamo il metodo *length* sull'oggetto **river**)
- L'oggetto funge da *parametro implicito* nell'invocazione del metodo
  - E' come passare a *length* il parametro *river*

# Parametri impliciti ed espliciti

- Parametri (espliciti): dati passati ad un metodo. Non tutti i metodi hanno parametri espliciti

```
System.out.println(greeting) ;  
greeting.length(); // senza parametri espliciti
```

- Parametro implicito: oggetto su cui è invocato il metodo

```
System.out.println(greeting) ;
```

# Valore restituito

- Un risultato che il metodo ha calcolato e che viene passato al metodo chiamante per essere utilizzato nella computazione di quest'ultimo

```
int n = greeting.length();  
        // n contiene valore restituito
```

# Esempio

- `replace` esegue un'operazione di ricerca e sostituzione in una stringa

```
river.replace("issipp", "our");  
// costruisce una nuova stringa ("Missouri")
```

- Questo metodo ha:
  - ❑ 1 parametro implicito: la stringa "**Mississippi**"
  - ❑ 2 parametri espliciti: le stringhe "**issipp**" e "**our**"
  - ❑ 1 valore restituito: la stringa "**Missouri**"

# Definizione di un metodo

- Specifica il tipo dei parametri espliciti e il valore di restituzione
- Tipo del parametro implicito = la classe corrente; non è scritto nella definizione del metodo
- Esempio nella classe String

```
public int length()  
    // return type: int  
    // no explicit parameter  
  
public String replace(String target, String replacement)  
    // return type: String;  
    // two explicit parameters of type String
```

# Definizione di un metodo

- `void` è usato per indicare che il metodo non restituisce alcun valore

```
public void println(String output) // in class PrintStream
```

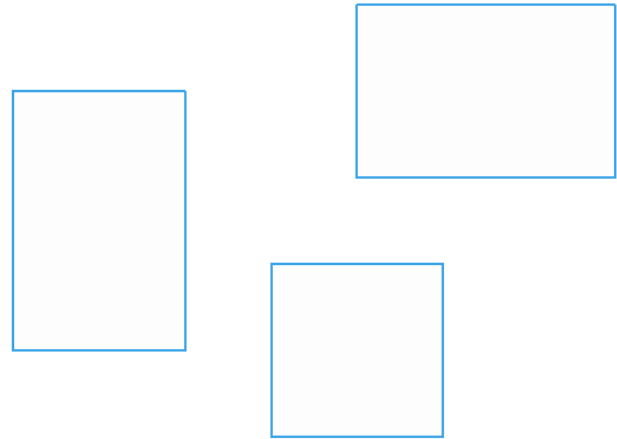
- Il nome di un metodo è sovraccaricato (overloaded) se ci sono più metodi con lo stesso nome nella classe

```
public void println(String output)  
public void println(int output)
```



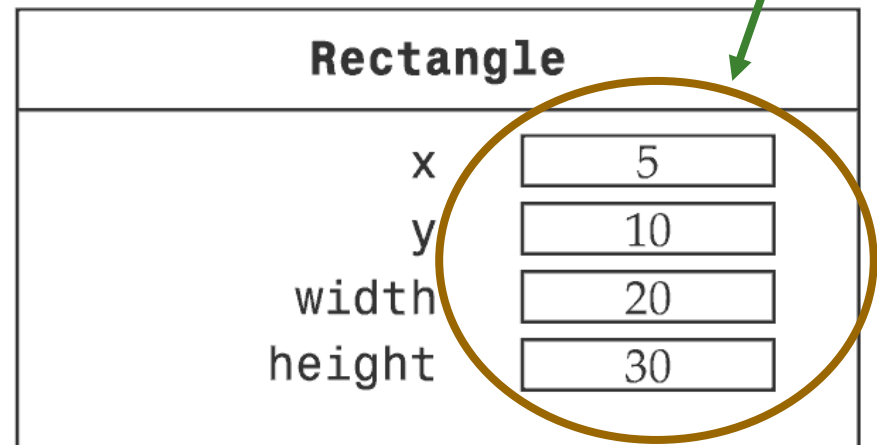
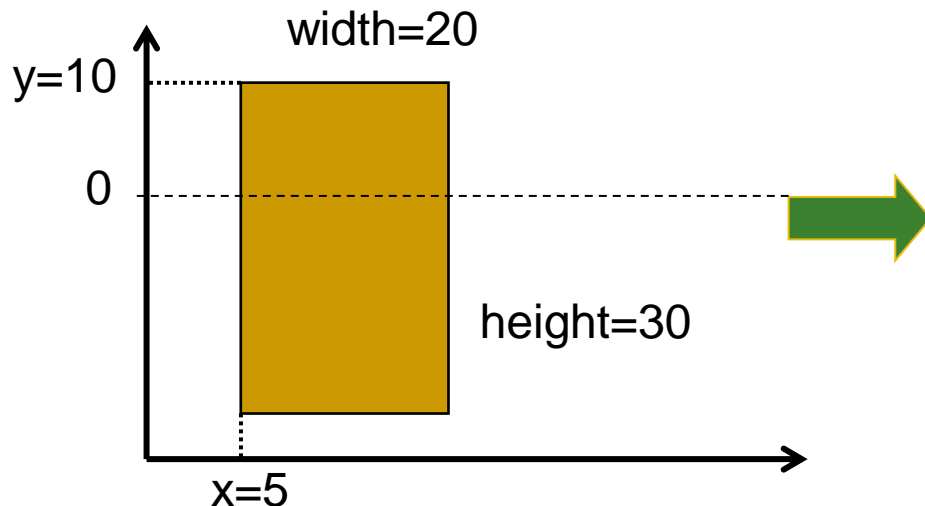
# Classe Rectangle

- Oggetti di tipo **Rectangle**:



- Per descrivere un rettangolo posso specificare

- L'ascissa **x** e l'ordinata **y** del suo angolo superiore sinistro
- il valore della larghezza (**width**)
- il valore dell'altezza (**height**)



# Classe Rectangle cont.

- Operazioni possibili:
  - ❑ traslazione del punto iniziale: `translate(x,y)`
  - ❑ recupero valore altezza: `getHeight()`
  - ❑ modifica ampiezza e altezza: `setSize(w,h)`
  - ❑ intersezione con altro rettangolo: `intersection(R)`
  - ❑ test intersezione non vuota: `intersects(R)`
  - ❑ test uguaglianza: `equals(O)`
  - ❑ etc.

# Classificazione metodi

- Metodi di accesso: non cambiano lo stato del parametro implicito

```
double width = box.getWidth();
```

- Metodi modificatori: cambiano lo stato del parametro implicito

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

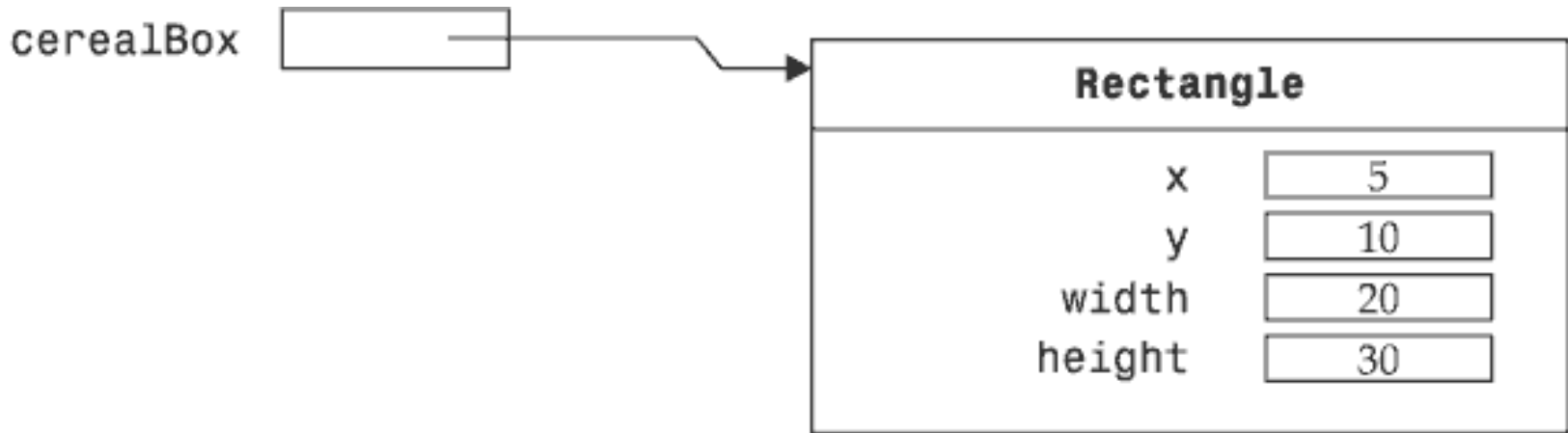
- Le classi di oggetti immutabili non contengono metodi modificatori

# Il metodo costruttore

```
public Rectangle(int x_init,int y_init,int width_init, int height_init) {  
    x=x_init;  
    y=y_init;  
    width=width_init;  
    height=height_init;  
}
```

- Una **classe** può implementare un **metodo** particolare, detto **costruttore**, che serve a inizializzare lo stato degli oggetti in fase di istanziamento
- Quando esiste, un **costruttore** deve chiamarsi come la **classe**
- Per creare un oggetto di una classe deve essere invocato un costruttore della classe in combinazione con l'operatore **new**
- Una variabile di istanza che non è inizializzata nel costruttore è inizializzata automaticamente a:
  - ❑ 0 se si tratta di un tipo numerico
  - ❑ null se si tratta di un riferimento

# Variabili che contengono oggetti



- La variabile `cerealBox` contiene un riferimento (indirizzo) ad un oggetto di tipo `Rectangle`.

# Creazione di oggetti (1)

- Rectangle cerealBox;

- Definisce una variabile oggetto Rectangle non inizializzata

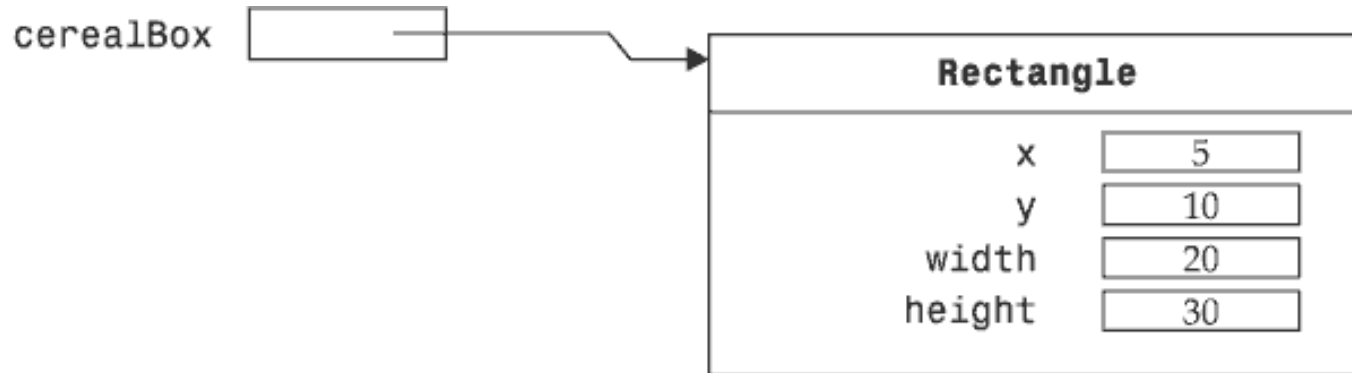
cerealBox



- è buona norma inizializzare sempre le variabili di tipo oggetto

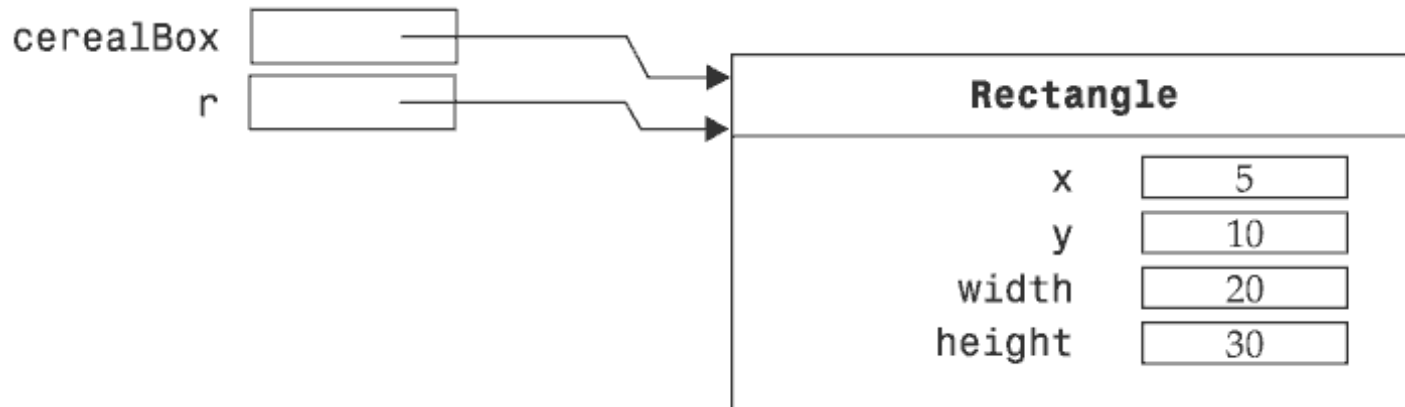
- cerealBox = new Rectangle(5, 10, 20, 30);

- Crea un rettangolo e assegna il suo indirizzo a cerealBox



# Creazione di oggetti (2)

- `Rectangle cerealBox = new Rectangle(5, 10, 20, 30);`
- `Rectangle r = cerealBox;` (assegnamento a variabile)
  - Si ottengono due variabili di tipo oggetto che si riferiscono allo stesso oggetto



# Creazione di Oggetti (3)

- Quando viene creato l'**oggetto** cerealBox con `Rectangle cerealBox = new Rectangle(5, 10, 20, 30);` viene allocato uno spazio di memoria in cui sono conservati:
  - i valori di `x`, `y`, `width` e `height`
    - ciascuna istanza di `Rectangle` ha le proprie copie delle variabili `x`, `y`, `width` e `height`
  - un riferimento alla classe `Rectangle` di appartenenza
    - il codice di `Rectangle` con tutti i suoi metodi (costruttore, `translate`, etc) è caricato nello spazio di memoria riservato alla classe (viene caricato in memoria una sola volta per tutti gli oggetti di tipo `Rectangle`)



# Creazione di Oggetti (4)

- Rectangle cerealBox =  
                                  new Rectangle(5, 10, 20, 30);
- Rectangle r = new Rectangle(5, 10, 20, 30);
  - Si definiscono due variabili inizializzate con due oggetti distinti di tipo Rectangle
  - r e cerealBox si riferiscono a due oggetti che sono indistinguibili rispetto allo stato (stesso stato) e al comportamento, ma hanno identità differenti

# Implementare un programma test

- Scrivi una nuova classe con il metodo `main`
- All'interno del metodo `main` costruisci uno o più oggetti
- Applica i metodi agli oggetti
- Importante:
  - **Copertura**: ogni istruzione dei metodi da testare devono essere eseguite almeno una volta
  - **Rieseguibilità**: il test deve poter essere rieseguito sotto le stesse condizioni (nessun input da operatore, dati presi da file o da codice)
  - **Tracciamento**: visualizzazione degli effetti delle operazioni eseguite (stato oggetti prima e dopo le operazioni, indicazione dell'operazione eseguita)

# Importazione pacchetti

Per usare le classi delle librerie o riutilizzare codice proprio può essere necessario importare delle classi:

- ❑ le classi Java sono raggruppate in pacchetti
- ❑ le classi di libreria si importano specificando il pacchetto e il nome della classe

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

- ❑ Non è necessario importare le classi del pacchetto `java.lang` come `String` e `System`
- ❑ Da Java 9.0, se si usano i moduli (creazione file `module-info.java`), in `module-info.java`:

```
requires java.desktop;
```

# File MoveTester.java

```
import java.awt.Rectangle;

public class MoveTester {
    public static void main(String[] args){
        Rectangle box = new Rectangle(5, 10, 20, 30);
        System.out.println("Rectangle top-left corner: (" +
                           box.getX() + ", " + box.getY() + ") " );

        // Move the rectangle
        box.translate(15, 25);

        // Print information about the moved rectangle
        System.out.println("After moving by 15 to the right
                           and by 25 downward, the top-left corner is:");
        System.out.println(box.getX());
        System.out.println(box.getY());
    }
}
```