Roadmap
parte 1

Roadmap

- Come è fatto un computer?
  - Architettura di Von Neumann Cap 1
- A cosa serve un computer?
  - Memorizzare ed elaborare dati (numerici, testuali, immagini, audio, video, etc) tramite algoritmi Cap 1
- Come si rappresentano i dati nel computer?
  - Rappresentazione dei dati APP D
- Noi usiamo Java, che tipi di dati mette a disposizione?
  - Tipi primitivi Cap 4
  - Come memorizzare e manipolare i dati? Definire variabili di tipi primitivi Cap 2

Roadmap

- Cosa posso fare con questi tipi di dati? Cap 4
  - Semplici operazioni su dati numerici
  - Operazioni messe a disposizione dal linguaggio:
    - la classe Math (statica, cioè è solo un contenitore di metodi)

- L'elaborazione di singoli caratteri è poco interessante, ci interessano di più le sequenze di caratteri: le stringhe! Cap4
  - Non sono un tipo primitivo ma oggetti creati dalla classe String

Roadmap

- Come posso acquisire dati dall'esterno del programma
  - Scanner Cap 4
- Per modellare realtà più complesse, ci sono le classi
  - Utilizzo di classi (definizione di variabili oggetto, invocazione di metodi statici e non) Cap2
- Come posso realizzare io una classe? Cap 3
  - Variabili di esemplare, metodi di accesso, metodi modificatori
- Finora gli esempi visti hanno avuto un flusso di istruzioni sequenziale. E' possibile alterarlo?
  - Decisioni: if, switch Cap 5
  - Cicli: while, for Cap 6

Utilizzo di Classi e Oggetti

Motivazioni

- Elaborando numeri e caratteri si possono scrivere programmi interessanti, ma ***programmi più utili*** hanno bisogno di manipolare ***dati più complessi***
  - numeri complessi, conti bancari, veicoli, negozi, dati anagrafici, forme grafiche, file audio, ecc.
- Il linguaggio Java gestisce questi dati complessi sotto forma di ***oggetti***
- Gli ***oggetti*** e il loro ***comportamento*** vengono descritti mediante le ***classi*** e i loro ***metodi***

Motivazioni

- Elaborando numeri e caratteri si possono scrivere programmi interessanti, ma ***programmi più utili*** hanno bisogno di manipolare ***dati più complessi***
  - numeri complessi, conti bancari, veicoli, negozi, dati anagrafici, forme grafiche, file audio, ecc.

- Java gestisce questi dati complessi sotto forma di ***oggetti***
  - *Rappresentazione di oggetti del mondo reali o loro astrazioni*
  - *Gli oggetti svolgono azioni che possono modificare il loro stato e/o influenzare altri oggetti tramite porzioni di codice detti* ***metodi***

Esempio: incrocio stradale

- *Simulazione di incrocio stradale per analizzare il flusso del traffico*
- *Il programma utilizzerà*
  - *Tanti oggetti, ognuno rappresentante una singola automobile che entra nell'incrocio*
  - *Oggetti per rappresentare le corsie*
  - *Oggetti per rappresentare i semafori*

- Le interazioni tra questi oggetti permettono di trarre conclusioni sulla situazione del traffico

Programmazione ad oggetti

- Paradigma di programmazione
  - Insieme di strumenti concettuali che caratterizzano un linguaggio di programmazione
- Nella programmazione ad oggetti si utilizzano le classi come "fabbriche di oggetti" o "stampi"
  - Tutti gli oggetti di una classe condividono le stesse *caratteristiche*
    - Automobile: targa, velocità attuale, livello carburante
  - Ciascun oggetto avrà un suo valore specifico di queste caratteristiche
    - Targa AC123DE, 30km/h, 20 litri di benzina
    - Targa FG456HI, 0km/h, 5 litri di benzina

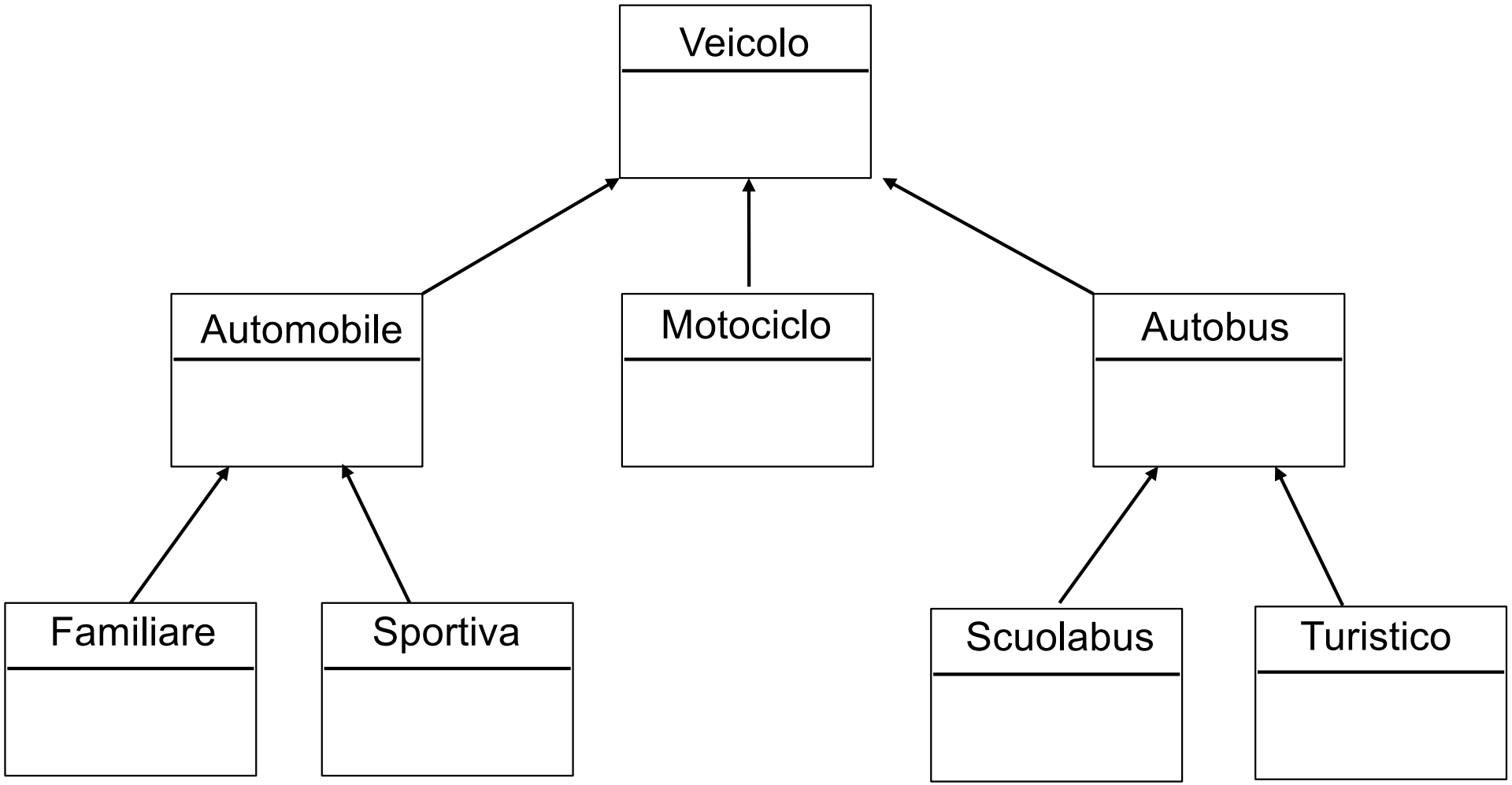
Programmazione ad oggetti

- Paradigma di programmazione
  - Insieme di strumenti concettuali che caratterizzano un linguaggio di programmazione
- Nella programmazione ad oggetti si utilizzano le classi come "fabbriche di oggetti" o "stampi"
  - Tutti gli oggetti di una classe condividono gli stessi *comportamenti*
    - Automobile: va avanti, va indietro, etc…
  - Ciascun oggetto potrà comportarsi in modo diverso
    - Targa AC123DE, va avanti
    - Targa FG456HI, sta ferma

Programmazione ad oggetti

- Incapsulamento
  - Racchiude gli oggetti in una "capsula" e ne rende accessibile/visibile solo una parte
  - Information hiding
- Polimorfismo
  - Permette a una stessa istruzione di avere significati differenti in base al contesti
    - Il metodo toString() restituirà una stringa con la descrizione degli attributi dell'oggetto che lo invoca (Persona vs Veicolo vs Animale)
- Ereditarietà
  - Modo per organizzare le classi definendo una sola volta attributi comuni creando una gerarchia

Esempio di gerarchia delle classi



OOP: Classi

- Una classe
  - Contiene dati e metodi la cui realizzazione non è necessariamente nota
  - Permette accesso e/o modifica di dati tramite la sua interfaccia pubblica
- Incapsulamento: diversi livelli di astrazione
  - In java è implementato con le classi
  - Black box

OOP: Classi

- Una ***classe***
  - è una ***fabbrica di oggetti*** o, meglio, uno **schema di progetto** per la costruzione di oggetti
    - gli oggetti che si creano sono *esemplari* (o "*istanze*", *instance*) di una classe
  - specifica **i metodi** che si possono invocare con gli oggetti che sono esemplari di tale classe (l'interfaccia pubblica)
  - definisce **i dettagli** della realizzazione dei metodi (codice e dati)
  - è anche un ***contenitore*** di
    - metodi statici (**Hello** contiene **main**)
    - variabili statiche (**System** contiene **out**)

Finora abbiamo visto solo questo aspetto, che è quello meno importante

OOP: oggetti e metodi

- Gli ***oggetti*** e il loro ***comportamento*** vengono descritti mediante le ***classi*** e i loro ***metodi***
  - Un ***oggetto*** è un'entità con determinate caratteristiche che può essere manipolata in un programma mediante l'invocazione di ***metodi***
  - ***Metodi:*** sequenze di istruzioni che possono accedere ai dati interni dell'oggetto e modificarli oppure visualizzarli

Obiettivi

- Iniziamo lo studio delle classi analizzando come si ***usano*** oggetti di una classe che si suppone **già progettata** da altri
  - Es: la classe String
    - Impareremo a conoscerla meglio più avanti. Per adesso ci interessa capire in generale come creare oggetti e invocare metodi usando come esempio oggetti String e metodi di questa classe
  - vedremo quindi che è possibile ***usare oggetti di cui non si conoscono i dettagli realizzativi ma solo l'interfaccia pubblica***
- **Usare oggetti che siano istanze di una classe** e **realizzare una classe** sono due attività ben distinte!

Interfaccia pubblica

- **L'interfaccia pubblica** di una classe descrive i metodi che possono essere invocati da oggetti "fabbricati" da quella classe (anche detti istanze o esemplari della classe).
- Ad esempio, alcuni metodi della classe String sono:
  - length() che restituisce la lunghezza della stringa
  - toUpperCase() che restituisce la stringa con tutte le lettere minuscole trasformate in maiuscolo
  - Come faccio a conoscerne altri? Niente paura! La documentazione della libreria standard ci aiuterà!

Definizione e inizializzazione di una variabile oggetto di tipo String

- Per invocare un metodo di una classe che fabbrica oggetti (come String) devo prima creare un oggetto di quella classe e inizializzarlo
- Per le stringhe posso dichiarare una variabile e usare l'assegnazione come si fa per i tipi primitivi (NB: solo per gli oggetti String posso fare questo tipo di inizializzazione)

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

  - Ora ho a disposizione una variabile oggetto di tipo String che si chiama greeting e che mi permette di rappresentare la stringa "Hello, World" e di poter effettuare delle elaborazioni su di essc

Invocazione di metodi

- Per invocare un metodo della classe String, ad esempio length(), devo utilizzare una variabile oggetto di tipo String che dica su che dati il metodo deve lavorare. La sintassi per invocare un metodo è:

```
nomeOggetto.nomeMetodo(parametri);
```

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

greeting.length();
```

Nome oggetto

Nome metodo

Metodi

- Istruzioni valide:
  - Se il metodo restituisce un valore di un certo tipo, posso assegnare questo valore ad una variabile di quello stesso tipo.
    - Ad esempio length() restituisce un intero

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

    - Ad esempio toUpperCase() restituisce una stringa

```
String river = "Mississippi";
String bigRiver = river.toUpperCase();
System.out.println(bigRiver); // stampa MISSISSIPPI
```

Metodi

- Istruzioni valide:
  - Se il metodo restituisce un valore di un certo tipo, posso utilizzare questo valore come parametro esplicito di un altro metodo.

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

    - Stamperà la lunghezza di "Hello, World!"

Metodi

- Istruzioni non valide:
  - Con un oggetto che è un'istanza di una data classe posso invocare solo metodi messi a disposizione da quella classe
  - out è un oggetto di tipo PrintStream. Questa classe non ha tra i suoi metodi il metodo length()

```
System.out.length();   // ERRORE
```

Metodi: parametri espliciti/impliciti

- Alcuni metodi necessitano di **valori in ingresso**, o **parametri espliciti**, che specifichino i dati da elaborare

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

  - greeting è un parametro esplicito: e' la stringa che voglio stampare
  - I parametri espliciti sono quelli tra parentesi

- Altri metodi no: tutte le informazioni necessarie sono memorizzate nell'oggetto corrispondente, il **parametro implicito**

```
int n = greeting.length();
```

  - Qui greeting è il parametro implicito: length viene calcolata sui dati contenuti nella variabile oggetto che invoca il metodo e perciò restituirà la lunghezza di "Hello, World!"

Un metodo con parametri espliciti e parametro implicito

```
public String replace(String oldStr, String newStr);
```

- Il metodo replace della classe String riceve due parametri espliciti, entrambi stringhe, e cerca nella stringa descritta dal parametro implicito le occorrenze del primo parametro per sostituirle con il secondo. Restituisce la stringa ottenuta in un altro oggetto, senza modificare la stringa associata al parametro implicito
  - Questa è una caratteristica di tutti i metodi di String, ma non è vera per tutte le classi. Esistono metodi, detti modificatori, che modificano i dati interni alla classe.

Un metodo con parametri espliciti e parametro implicito

```
river.replace("issipp","our");
```

- Due parametri espliciti: le stringhe "issipp" e "our"
- Un parametro implicito: l'oggetto river (la stringa "Mississippi")
- Un valore restituito: la stringa "Missouri"
  - Mississippi => Missouri
  - ATTENZIONE: per quanto detto nella slide precedente river contiene ancora "Mississippi"!

```
// stampa Missouri
System.out.println(river.replace("issip","our"));

System.out.println(river); // stampa Mississippi
```

Definizioni di metodi: firma

- Come faccio a sapere se il metodo che voglio utilizzare ha dei parametri espliciti, di che tipo sono, e in che ordine li devo specificare?
- Come faccio a sapere che tipo di dato restituisce un metodo?
- Queste informazioni sono contenute nella documentazione della classe, in quella che si chiama firma del metodo

```
public String replace(String target, String replac);
```

Tipo restituito

Nome del metodo

Parametri espliciti

Definizioni di metodi: firma

```
public void println(String output);
public String replace(String target, String replac);
```

- La ***definizione di un metodo*** inizia con la sua **intestazione** (**firma, signature**):
  - uno **specificatore di accesso**
    - Indica quali altri metodi possono invocare il metodo
      - Un metodo **public** può essere invocato da qualsiasi altro metodo di qualsiasi altra classe
      - Un metodo può essere anche **private**
  - il tipo di dati **restituito** dal metodo (**String, void, int, ...**)
  - il **nome** (identificatore) del metodo (**println, replace, ...**)
  - un **elenco di parametri espliciti,** eventualmente vuoto, tra **parentesi tonde**
    - di ogni parametro si indica il tipo ed il nome
    - più parametri sono separati da una virgola

Definizioni di metodi: firma

- La dichiarazione di un metodo specifica il tipo di dati restituito al termine della sua invocazione

```
int n = river.length();
```

  - Il metodo length() restituisce un valore int

- Se un metodo non restituisce nessun valore si dichiara il tipo speciale void.
  - Il valore restituito, se assegnato, deve essere assegnato ad un tipo di dati corrispondente

```
double b = System.out.println(river); //ERRORE
System.out.println(river);            //OK
```

Esercizio

- Si consideri la seguente invocazione su un oggetto river di tipo String: river.length()
  - Parametri impliciti?
    - river
  - Parametri espliciti?
    - nessuno
  - Tipo di valore restituito?
    - int

Variabili oggetto

CODICE
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Variabili di tipo primitivo e variabili oggetto

- Abbiamo visto che per gestire i tipi primitivi devo dichiarare una variabile di quel tipo e poi assegnarle un valore compatibile
  - Quando uso la variabile, uso il suo valore

```
int a = 10;
System.out.println(a); // Stampa 10
int b = a + 3;  // b contiene il valore 13
```

- Abbiamo visto che per invocare metodi di una classe devo creare una variabile oggetto di quella classe: cosa contiene quella variabile?

Variabili oggetto

- ***Una variabile oggetto*** conserva non l'oggetto stesso, ma informazioni sulla sua posizione nella memoria del computer
  - È detta anche ***riferimento*** o ***puntatore***
- Come già visto per oggetti della classe String, per definire una variabile oggetto si indica il nome della classe ai cui oggetti farà riferimento la variabile, seguito dal nome della variabile stessa

```
NomeClasse nomeOggetto;
```

ATTENZIONE: la definizione di una variabile oggetto crea un riferimento NON INIZIALIZZATO (= non fa riferimento ad alcun oggetto)

Costruire oggetti: l'operatore new

- ***A differenza di quanto*** visto per oggetti della classe String, in generale per ***creare un nuovo oggetto*** di una classe si usa l'***operatore*** **new** seguito dal ***nome della classe*** e da una coppia di parentesi tonde

```
new NomeClasse(parametri);
```

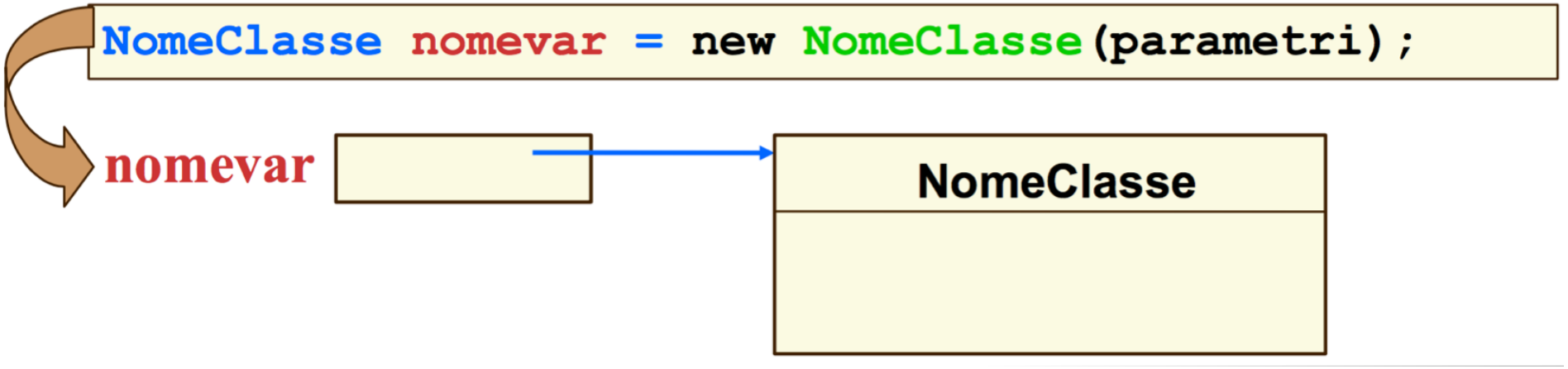
  - Questa inizializzazione è valida anche per String, ma meno usata.

```
String river = "Mississippi";
```

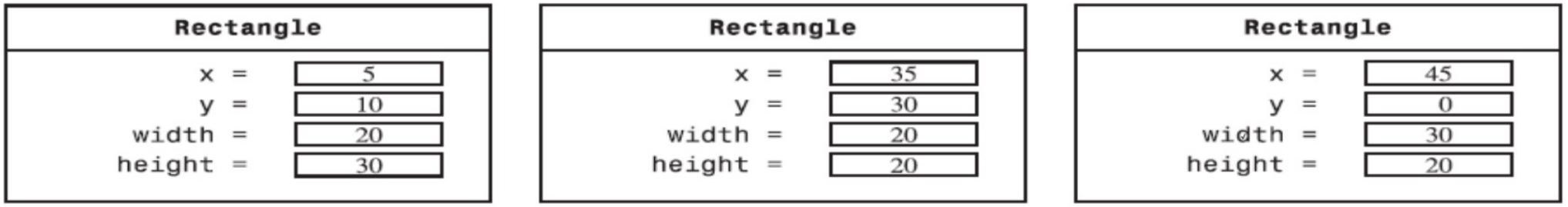
```
String river = new String("Mississippi");
```

Costruire oggetti: l'operatore new

- L'operatore **new** ***crea un nuovo oggetto e ne restituisce un riferimento***, che può essere assegnato a una variabile oggetto del tipo appropriato



Esempio: la classe Rectangle



- Un rettangolo è descritto dalle coordinate (x,y) del suo vertice in alto a sinistra, e da larghezza e altezza.
  - Per creare un rettangolo bisogna
  - Specificare x, y, width, height
  - Invocare l'operatore **new e il costruttore con i parametri richiesti**
  - **Assegnare il rettangolo appena creato ad una variabile oggetto**

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

Costruttori vs metodi

- Il processo di creazione di un nuovo oggetto è detto costruzione e ciò che segue l'operatore new è detto **costruttore**
- Un costruttore non è un metodo
  - Ci assomiglia perché il nome è seguito dalle parentesi e può avere dei parametri espliciti
  - Il suo nome è uguale a quello della classe
    - Inizia con la maiuscola!
  - Non viene invocato da un parametro implicito
    - Utilizzabile solo con new

```
Rectangle box = new Rectangle(5,10,20,30);
box.Rectangle(20,35,20,30);   // ERRORE
```

Costruttori vs metodi

- NB: Se voglio re-inizializzare un oggetto devo crearne uno nuovo (con new) e assegnarne il riferimento alla variabile che sto utilizzando

```
Rectangle box = new Rectangle(5,10,20,30);
box = new Rectangle(20,35,20,20);  // OK
```

  - Ora box contiene un riferimento ad un oggetto che si trova in posizione x=20, y=35 e ha dimensioni 20 e 20

Metodi della classe Rectangle

- Per vedere che metodi ha a disposizione un oggetto istanza della classe Rectangle basta cercare la classe nella documentazione Java
  - Ad esempio trovero' il metodo
    - void translate(int dx, int dy) che trasla della quantita' dx la coordinata x, e della quantita' dy la coordinata y
    - double getX() restituisce il valore della coordinata x

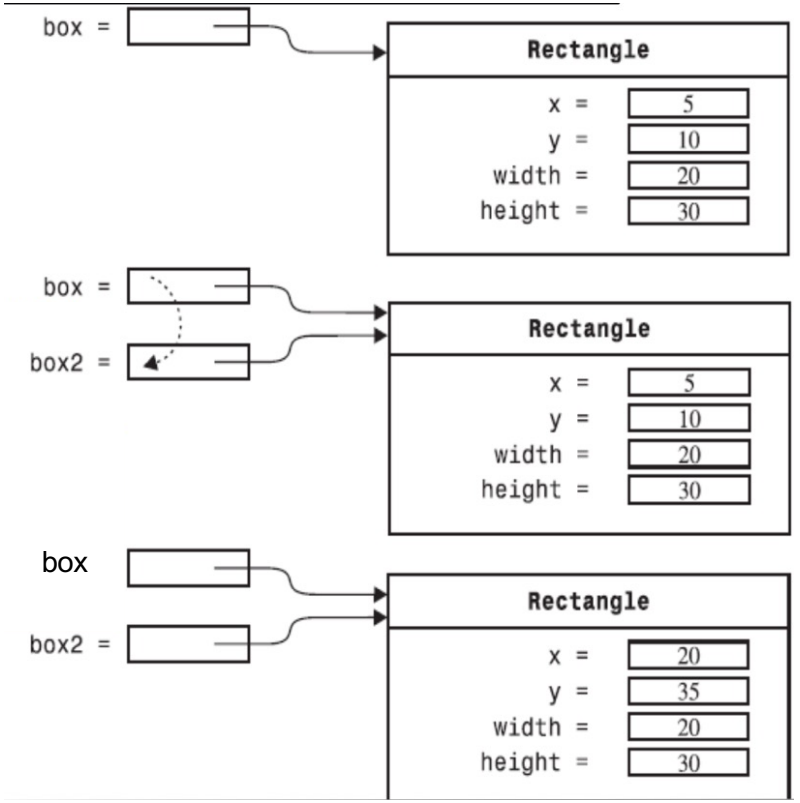
```
Rectangle box = new Rectangle(5,10,20,30);
System.out.println(box.getX()); // stampa 5
box.translate(10,10);
System.out.println(box.getX()); // stampa 15
```

Copiare oggetti

Riferimenti ad oggetti

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15,25);
System.out.println(box.getX());  // output 20
System.out.println(box2.getX()); // output 20
```

- Attenzione: se assegno direttamente l'oggetto di fatto sto copiando il suo contenuto, ovvero il riferimento all'area di memoria!



Copiare variabili

- Per ottenere anche con variabili oggetto lo stesso effetto dell'assegnazione di variabili di tipi numerici fondamentali, è necessario ***creare esplicitamente una copia dell'oggetto con lo stesso stato***
  - ***Creo un nuovo oggetto con new*** inizializzandolo adeguatamente
  - Assegno il riferimento ad un'altra variabile oggetto

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = new Rectangle(box);
box2.translate(15,25);
System.out.println(box.getX());  // output 5
System.out.println(box2.getX()); // output 20
```

Copiare variabili

- Le **variabili di tipi numerici fondamentali** indirizzano una posizione in memoria che contiene un ***valore***, che viene copiato con l'assegnazione
  - cambiando il valore di una variabile, non viene cambiato il valore dell'altra
- Le **variabili oggetto** indirizzano una posizione in memoria che, invece, contiene un ***riferimento ad un oggetto***, e solo tale riferimento viene copiato con l'assegnazione
  - modificando lo stato dell'oggetto, tale modifica è visibile da entrambi i riferimenti
  - ***non viene creata una copia dell'oggetto!***

Collaudo di una classe

- Passi per costruire un programma di collaudo
  - Definire una nuova classe
  - Definire in essa il metodo main
  - Costruire oggetti all'interno di main
  - Applicare metodi agli oggetti
  - Visualizzare risultati delle invocazioni dei metodi

Esempio

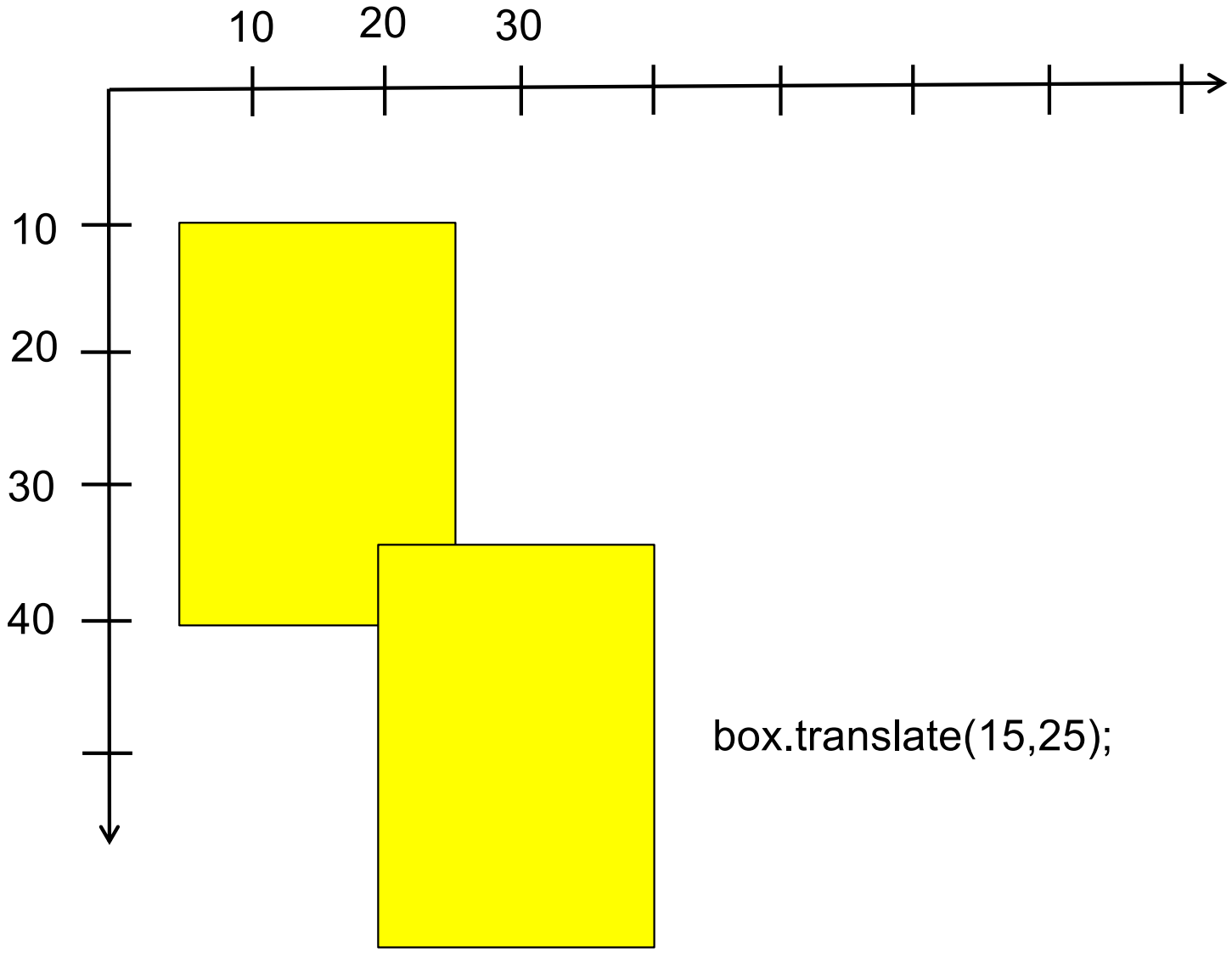
```
import java.awt.Rectangle;
public class MoveTester
{
   public static void main(String[] args)
   {
      Rectangle box = new Rectangle(5, 10, 20, 30);

      // sposta il rettangolo
      box.translate(15, 25);

      // visualizza informaz. su rettangolo traslato
      System.out.println("After moving, the top-left
            corner is:");
      System.out.println(box.getX());
      System.out.println(box.getY());
   }
}
```

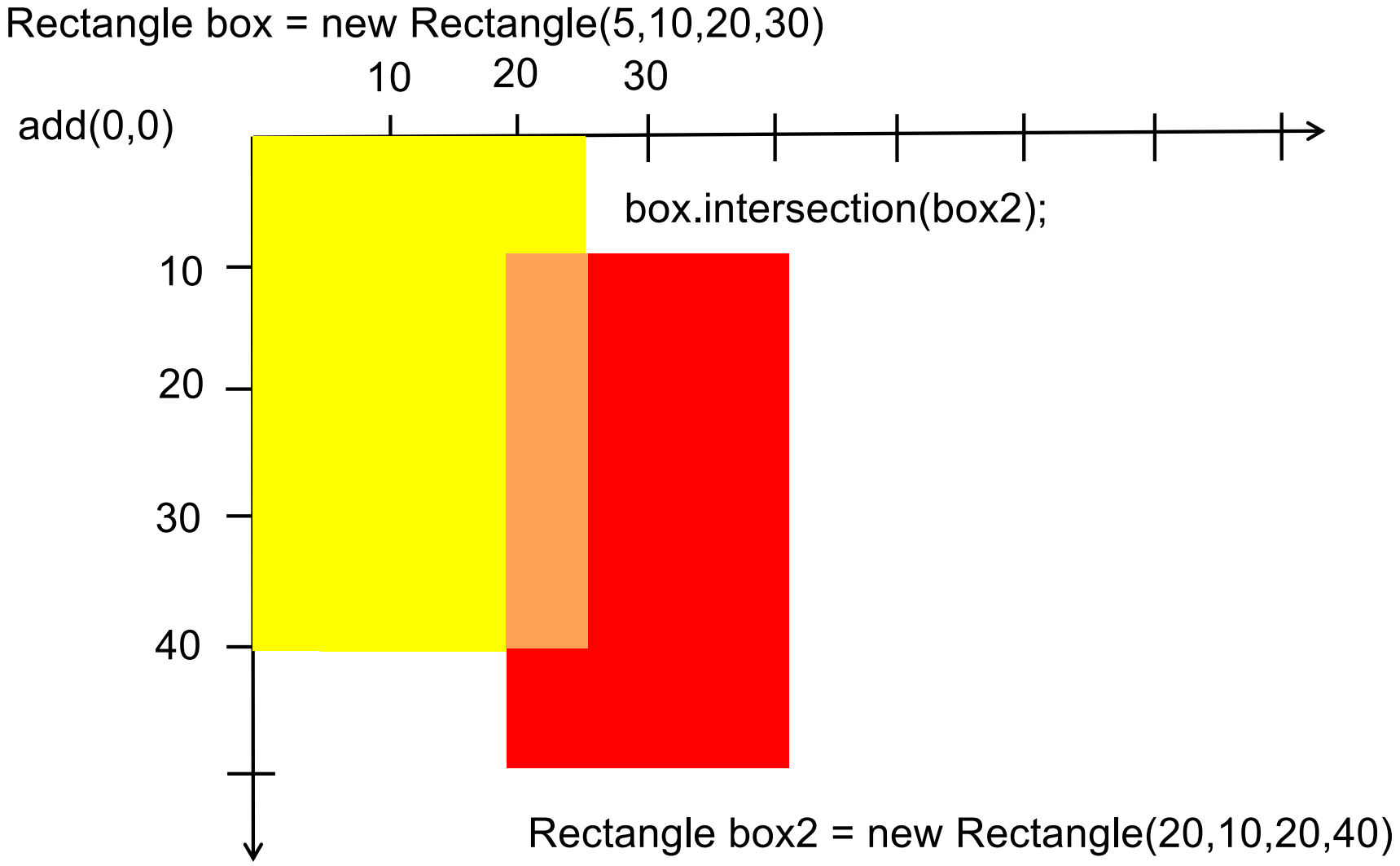
Esempio translate

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



box.translate(15,25);

Esempio add e intersection



Usare una classe BankAccount

- BankAccount()
  - crea un conto bancario inizialmente vuoto
- double getBalance()
  - Restituisce il valore del saldo
- void withdraw(double amount)
  - Preleva l'ammontare indicato come parametro esplicito
- void deposit(double amount)
  - Deposita l'ammontare indicato come parametro esplicito

Esercizio

- Scrivi una classe eseguibile BankAccountTester in cui
  - Creare una variabile oggetto di tipo BankAccount e la chiami account
  - Stampare il valore attuale del saldo di account
  - Depositare 100 euro in account
  - Stampare il valore attuale del saldo di account
  - Prelevare 20 euro da account
  - Stampare il valore attuale del saldo di account

Ora tocca a voi

- Scrivere una classe eseguibile CalcolaInteressi in cui si deve:
  - Creare una variabile oggetto di tipo BankAccount e chiamarla account
  - Depositare 100 euro
  - Calcolare un interesse del 5% sul valore attuale del saldo (Attenzione: devo conoscere il saldo, che metodo dovrò invocare?)
  - Depositare gli interessi sul conto
  - Stampare il valore attuale del saldo

Take home message

- Definire e utilizzare variabili
  - Contenenti un valore di tipo primitivo
  - Contenenti un riferimento ad un oggetto istanza di una data classe
- Invocare i metodi di una classe che "fabbrica" oggetti
  - Parametro implicito/parametri espliciti
  - oggetto.nomeMetodo(parametri);
- Invocare i metodi di una classe che contiene metodi statici
  - Solo parametri espliciti
  - NomeClasse.nomeMetodo(parametri);
- Consultare la documentazione standard, importare pacchetti, realizzare programmi di collaudo