Le costanti statiche

- Una *costante statica* è dichiarata usando le parole chiave **`static`** e **`final`**
  - **Es.:** `public static final COSTO_COMMISS=1.5;`
- E` ragionevole dichiarare statica una costante
  - Sarebbe inutile che ciascun oggetto della classe **`BankAccount`** avesse una propria variabile **`COSTO_COMMISS`** con valore costante 1.5
  - E` molto meglio che tutti gli oggetti della classe **`BankAccount`** facciano riferimento ad un'unica variabile **`COSTO_COMMISS`**
- Le costanti statiche si possono usare liberamente

Visibilità delle variabili

- Campo di visibilità di una variabile (scope): parte del programma in cui si può fare riferimento alla variabile mediante il suo nome
- Campo di visibilità di una variabile locale: dalla sua dichiarazione alla fine del blocco
  - Nell'ambito di visibilità di una variabile locale non è possibile definirne un'altra avente lo stesso nome (nomi non si possono ridefinire in blocchi annidati)
  - Esempio:

```
for (int i=0; i<10; i++){
    …
    float i = 3.5;
     /* errore: qui non si può dichiarare
        un'altra variabile di nome i */
}
```

Visibilità sovrapposte

- I campi di visibilità di una variabile locale e di una variabile di istanza possono sovrapporsi
- La variabile locale oscura la variabile di istanza con lo stesso nome

```
public class Coin
{
    public void draw(Graphics2D g2)
    {
      String name = "SansSerif"; // variabile locale
      ...
    }
    private String name; //variabile di istanza
    private double value;
}
```

Visibilità sovrapposte

- Se in un metodo si vuole fare riferimento ad una variabile di istanza che ha lo stesso nome di una variabile locale allora occorre usare il riferimento **`this`**

```
public class Coin
{
   public void draw(Graphics2D g2)
   {
    String name = "SansSerif"; // variabile locale
    g2.setFont(new Font(name, . . .)); /* qui name si riferisce
                                       alla variabile locale */
    g2.drawString(this.name, . . .); /* qui name si riferisce alla
                                        variabile di istanza */
   }
   private String name; // variabile di istanza
   . . .
}
```

Visibilità sovrapposte

- Errore tipico nei costruttori

```
public class Coin{
   public Coin(double inBalance, String aName)
   {
     String name = aName; // variabile locale, non di istanza
     balance = inBalance;
   }
   ...
   private String name; // variabile di istanza
   private double balance; // variabile di istanza
}
```

Visibilità di membri di classe

- All'interno di una classe si può accedere alle variabili di istanza e ai metodi della classe specificandone semplicemente il nome (si sottintende il parametro implicito o il nome della classe stessa come prefisso)

Esempio:

```
public void trasferisci(double somma, BankAccount altro)
 {
  preleva(somma); // equivale a this.preleva(somma)
  altro.deposita(somma)  ;
 }
```

Pacchetti

- Insieme di classi correlate
- Libreria Java costituita da numerosi package
- Possibile dichiarare appartenenza di una classe ad un package mettendo sulla prima riga del file che contiene la classe:

```
package packagename;
```

  - **Esempio :**

```
package com.horstmann.bigjava;
public class Numeric
{
  ...
}
```

  - Se la dichiarazione è omessa, le classi create fanno parte di un package di default (senza nome)

Alcuni pacchetti della libreria Java

Package	Scopo	Classi campione
java.lang	Supporto al linguaggio	Math
java.util	Utility	Random
java.io	Input/output	PrintStream
java.awt	Abstract Windowing Toolkit (Interfacce grafiche)	Color
java.applet	Applet	Applet
java.net	Connessione di rete	Socket
java.sql	Accesso a Database	ResultSet
javax.swing	Interfaccia utente Swing	JButton
org.omg.CORBA	Common Object Request Broker Architecture	ORB

Nomi dei pacchetti

- E' necessario un meccanismo che garantisca l'unicità dei nomi delle classi e dei package.
- Difficile pensare di usare nomi di classi differenti
- Basta assicurarsi che i nomi dei package siano differenti
- Per convenzione i nomi dei **package** sono scritti in lettere **minuscole**

Nomi dei pacchetti

- Per rendere unici i nomi dei pacchetti si possono usare i nomi dei domini Internet alla rovescia
  - Esempi: `it.unisa.mypackage`
    `com.horstmann.bigjava`

- In generale una persona non è l'unico utente di un dominio Internet, quindi meglio usare l'intero indirizzo di e-mail.
  - **Esempio :**
    rossi@dmi.unisa.it diventa `it.unisa.dmi.rossi`

Pacchetti e posizione nel file system

- Il nome del pacchetto deve coincidere con il percorso della sottocartella dove è ubicato il pacchetto
  - **Esempio:** il pacchetto **com.horstmann.bigjava** deve essere ubicato nella sottocartella: **com/horstmann/bigjava**
    - Il percorso della sottocartella è specificato a partire da una directory prefissata o dalla directory corrente

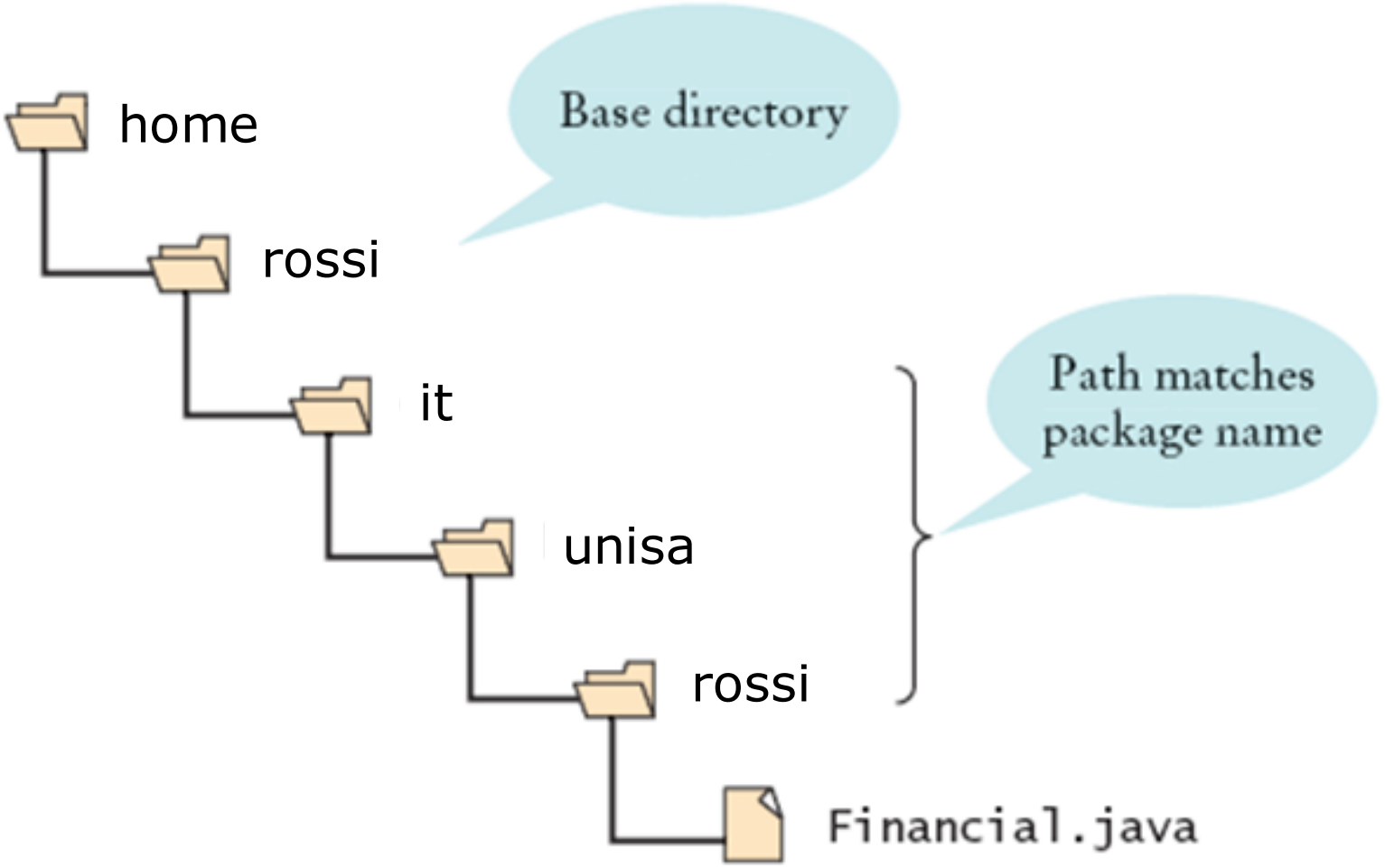
Localizzazione dei pacchetti

- Supponiamo che la directory corrente sia /home/rossi e che in un file (con estensione java) vogliamo importare il package it.unisa.rossi

- I file che compongono il package devono stare nella sottodirectory it/unisa/rossi della directory corrente, cioè in

/home/rossi/it/unisa/rossi

Cartella di base e sottocartelle per i pacchetti



Localizzazione dei pacchetti

- Se vogliamo che Java cerchi i file componenti un package a partire da una particolare directory, possiamo
  - assegnare il suo path assoluto alla variabile di ambiente CLASSPATH
    - Es. export CLASSPATH=/home/rossi/esercizi: (UNIX)
    - Tutte le volte che importo classi non standard la ricerca parte da /home/rossi/esercizi
    - Comodo ma **non garantito** su tutti i sistemi e/o tutte le installazioni del JDK
  - Usare l'opzione -classpath del compilatore javac (garantito)

```
javac -classpath /home/rossi/esercizi Numeric.java
```

Importare pacchetti

- Si può sempre usare una classe senza importarla
  - **Esempio:**

```
java.awt.Rectangle r
        = new java.awt.Rectangle(6,13,20,32);
```

- Per evitare di usare nomi qualificati possiamo usare la parola chiave import
  - **Esempio:**

```
import java.awt.Rectangle;
. . .
Rectangle r = new Rectangle(6,13,20,32);
```

Importare pacchetti

- Si possono importare tutte le classi di un pacchetto
  - **Esempio:**

```
import java.awt.*;
```

- Nota: non c'è bisogno di importare java.lang per usare le sue classi

Il Problema della Collisione

- Se importiamo due package che contengono entrambi una certa classe *Myclass*, un riferimento a *Myclass* nel codice genera una collisione sul nome *Myclass.*
- In questo caso il compilatore chiede di usare i nomi completi per evitare ambiguità.
- Dati i package *pack1* e *pack2*, ci riferiremo alle classi *Myclass* come

  `pack1.Myclass` e **`pack2.Myclass`**

Il significato di import

- L'istruzione **import** dice soltanto al compilatore dove si trova un certo package o una certa classe.

- Per ogni riferimento ad una classe *Myclass*, che non faccia parte dello stesso package del file che stiamo compilando, il compilatore controlla **solo** l'esistenza del file *Myclass.class* nella locazione specificata da **import.**

Caricamento di Classi Importate

- Le classi importate, tramite l'istruzione **import** o specificando il loro nome completo, vengono caricate dal **Class Loader** a runtime

- Finché il codice non fa un riferimento esplicito ad una classe che è stata importata, la classe non viene caricata

Differenze tra import e #include

- **#include** del C e del C++
  - è una direttiva al preprocessore per inserire all'interno del sorgente un file contenente
    - prototipi delle funzioni di libreria e costanti predefinite oppure
    - prototipi di funzioni e costanti definite dal programmatore
  - Bisogna utilizzarla per forza
- **import** di java
  - È una semplificazione per specificare il nome di una classe
    - Non include niente nel file sorgente, dice solo dove si trova la classe
  - È possibile non usarla mai