

Progettare una classe



Individuare una classe

Una classe rappresenta un singolo concetto dal dominio del problema.

Il nome di una classe dovrebbe essere un sostantivo che descrive il concetto.

- Concetti dalla matematica:

- Punto
- Rettangolo
- Ellisse

- Concetti dalla vita reale:

- Conto Bancario
- CashRegister

Individuare una classe

- Attori (fine in -er, -or) — gli oggetti fanno alcuni tipi di lavoro per te:
 - Scanner
 - Random
- Classi di utilità — nessun oggetto, solo metodi statici e costanti:
 - Math
- Avviatori di programmi: una classe con solo un metodo principale
 - Il nome della classe dovrebbe indicare cosa faranno gli oggetti della classe: Paycheck è un nome migliore di PaycheckProgram.
 - Non trasformare una singola operazione in una classe: Paycheck è un nome migliore di ComputePaycheck.

Metodi

- Una classe dovrebbe rappresentare un singolo concetto.
- L'interfaccia pubblica di una classe è coesa se tutte le sue caratteristiche sono correlate al concetto che la classe rappresenta.
- I membri di una squadra coesa hanno un obiettivo comune.

Metodi

Questa classe manca di coesione (contiene due concetti, monete e pagamento):

```
public class CashRegister
{
    public static final double QUARTER_VALUE = 0.25;
    public static final double DIME_VALUE = 0.1;
    public static final double NICKEL_VALUE = 0.05;
    . . .
    public void receivePayment(int dollars, int
        quarters, int dimes, int nickels, int pennies)
    . . .
}
```

Metodi

```
public class Coin
{
    public Coin(double aValue, String aName) { . . . }
    public double getValue() { . . . }
    . . .
}
```

```
public class CashRegister
{
    . . .
    public void receivePayment(int coinCount, Coin coinType)
    {
        payment = payment + coinCount * coinType.getValue();
    }
    . . .
}
```

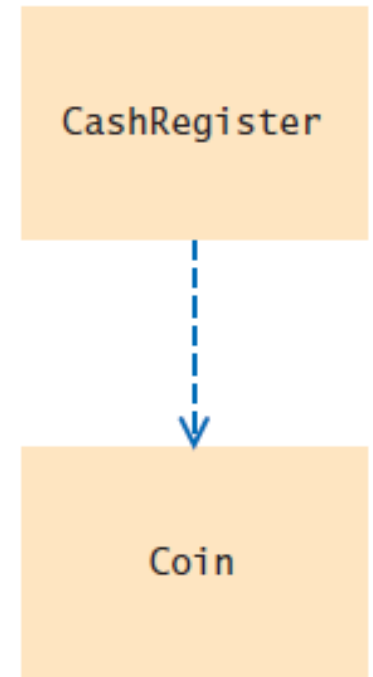
Minimizzare le dipendenze



Una classe dipende da un'altra classe se i suoi metodi usano quella classe in qualche modo.



CashRegister dipende da Coin, non viceversa



Accessi e modifiche

Un metodo mutator cambia lo stato di un oggetto.

Un metodo di accesso chiede a un oggetto di calcolare un risultato, senza modificare lo stato.

Una classe immutabile non ha metodi mutatori.

String è una classe immutabile

Nessun metodo nella classe String può modificare il contenuto di una stringa.

I riferimenti agli oggetti di una classe immutabile possono essere condivisi in modo sicuro.

Accessi e modifiche

In una classe mutabile, funzioni di accesso e mutatori separati

Un metodo che restituisce un valore non deve essere un mutatore.

In generale, tutti i mutatori della tua classe dovrebbero avere il tipo restituito vuoto.

A volte un metodo mutatore può restituire un valore informativo.

Il metodo di remove ArrayList restituisce true se la rimozione è riuscita.

Per controllare la temperatura dell'acqua nella bottiglia, potresti prendere un sorso, ma sarebbe l'equivalente di un metodo mutatore.

Effetto collaterale

- Un effetto collaterale di un metodo è qualsiasi modifica dei dati osservabile esternamente.
- I metodi mutatori hanno un effetto collaterale, ovvero la modifica del parametro implicito.
- In generale, un metodo non dovrebbe modificare le sue variabili di parametro.

```
/**  
Computes the total balance of the given accounts.  
@param accounts a list of bank accounts  
*/  
public double getTotalBalance(ArrayList<String> accounts)  
{  
    double sum = 0;  
    while (studentNames.size() > 0)  
    {  
        BankAccount account = accounts.remove(0); //NO  
        sum = sum + account.getBalance();  
    }  
    return sum;  
}  
}
```

A photograph of a collaborative workspace. Several people are gathered around a wooden table, their hands visible as they work. The table is covered with various documents, including one with the word 'INFORMATION' at the top. Numerous colorful sticky notes (pink, yellow, green) are placed across the papers. A color calibration chart is also visible. The scene is dimly lit, with a desk lamp providing light on the right side. The overall atmosphere is one of focused teamwork and creative problem-solving.

Problem solving:

progettare i dati degli oggetti

Gestione di un totale

Tutte le classi che gestiscono un totale seguono lo stesso schema di base.

Mantieni una variabile di istanza che rappresenti il totale corrente:

```
private double totale;
```

Fornire questi metodi se necessario:


Un metodo per aumentare il totale di un determinato importo

```
public void deposito(double amount)

{

    totale = totale + amount;

}
```




Gestione di un totale

Un metodo che riduce o cancella il totale

```
public void clear()  
{  
    totale = 0;  
}
```

Un metodo che restituisce il totale corrente

```
public double getTotale()  
{  
    return totale;  
}
```



Conteggio eventi

Un contatore che conta gli eventi viene incrementato nei metodi che corrispondono agli eventi.

ADV: Tieni un contatore:

```
private int contatore;
```

Incrementa il contatore in quei metodi che corrispondono agli eventi che vuoi contare:

```
public void deposito(double amount)
{
    totale = totale +
    amount;
    contatore ++;
}
```


Conteggio eventi

Fornire un metodo per azzerare il contatore, se necessario:

```
public void clear()  
{  
    totale = 0;  
    contatore = 0;  
}
```

Potrebbe essere necessario un metodo per segnalare il conteggio all'utente della classe.



Gestione raccolta valori

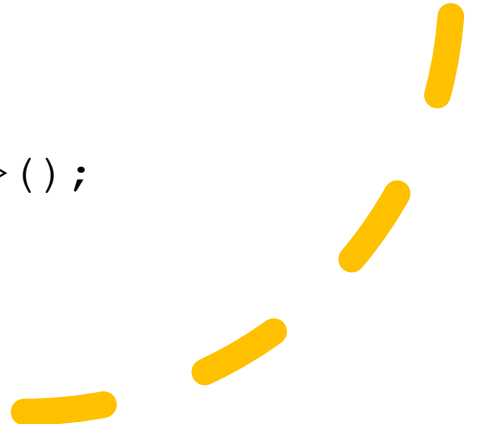
Un oggetto può raccogliere altri oggetti in un array o in un arraylist.

Esempio: Un oggetto carrello deve gestire una raccolta di articoli.

```
public class Question
{
    private ArrayList<String> choices;
    . . .
}
```

Inizializza la variabile di istanza su una raccolta vuota

```
public Question()
{
    choices = new ArrayList<String>();
}
```



Gestione raccolta valori

Fornire un meccanismo per aggiungere valori

```
public void add(String option)
{
    choices.add(option);
}
```



Gestione delle proprietà di un oggetto

Una proprietà è un valore caratteristico di un oggetto che può essere ispezionato e/o modificato dall'utente.

ADV: Fornisci una variabile di istanza per memorizzare il valore della proprietà e i metodi per ottenerla e impostarla.

```
public class Student
{
    private String name;
    . . .
    public String getName() {
        return name;
    }
    public void setName(String newName) {
        name = newName;
    }
    . . .
}
```

Gestione delle proprietà di un oggetto

```
public void setName(String newName)
{
    if (newName.length() > 0) {
        name = newName;
    }
}
```

Alcune proprietà non dovrebbero cambiare dopo essere state impostate nel costruttore

```
public class Student{

    private int id;
    . . .

    public Student(int anId) { id = anId; }

    public String getId() { return id; }

    // No setId method

    . . .
}
```

Oggetti con stati diversi

Alcuni oggetti hanno un comportamento che varia a seconda di ciò che è accaduto in passato.

Se un pesce è affamato, il suo comportamento cambia.

Fornire una variabile di istanza per lo stato corrente

```
public class Fish
{
    private int hungry;
    . . .
}
```

```
public static final int
NOT_HUNGRY = 0;
public static final int
SOMEWHAT_HUNGRY = 1;
public static final int
VERY_HUNGRY = 2;
```



Oggetti con stati diversi

Determina quali metodi cambiano lo stato:

```
public void eat()  
{  
    hungry = NOT_HUNGRY;  
    . . .  
}  
  
public void move()  
{  
    . . .  
    if (hungry < VERY_HUNGRY) {  
        hungry++;  
    }  
}
```



Oggetti con stati diversi

Determina dove lo stato influenza il
comportamento:

```
public void move()  
{  
    if (hungry == VERY_HUNGRY)  
    {  
        ....  
    }  
    . . .  
}
```



Descrizione della posizione di un oggetto

Per modellare un oggetto in movimento:

È necessario memorizzare e aggiornare la sua posizione.

Potrebbe anche essere necessario memorizzarne l'orientamento o la velocità.

Se l'oggetto si sposta lungo una linea, puoi rappresentare la posizione come distanza da un punto fisso:

```
private double distanzaDallaFine;
```

Se l'oggetto si sposta in una griglia, ricorda la posizione e la direzione correnti nella griglia:

```
private int row;
```

```
private int column;
```

```
private int direction;
```

```
// 0 = North, 1 = East, 2 = South, 3 = West
```



Descrizione della posizione di un oggetto

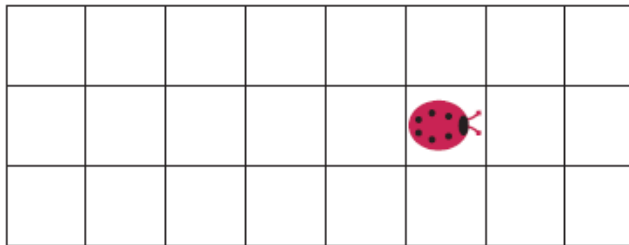
Una coccinella in una griglia deve memorizzare la sua riga, colonna e direzione.

Ci saranno metodi che
aggiorneranno la posizione.

Ti potrebbe essere detto

quanto si muove l'oggetto:

```
public void move(double distanzaPercorsa)
{
    distanzaDallaFine = distanzaDallaFine +
    distanzaPercorsa;
}
```



Descrizione della posizione di un oggetto

Se lo spostamento avviene in una griglia, è necessario aggiornare la riga o la colonna, a seconda dell'orientamento corrente

```
public void moveOneUnit()  
{  
    if (direction == NORTH) {  
        row--;  
    } else if (direction == EAST) {  
        column++;  
    } else if (direction == SOUTH) {  
        row++;  
    } else if (direction == WEST) {  
        column--;  
    }  
}
```

Variabili statiche e metodi statici

Una variabile statica appartiene alla classe, non a un oggetto della classe.

Variabili statiche e metodi statici

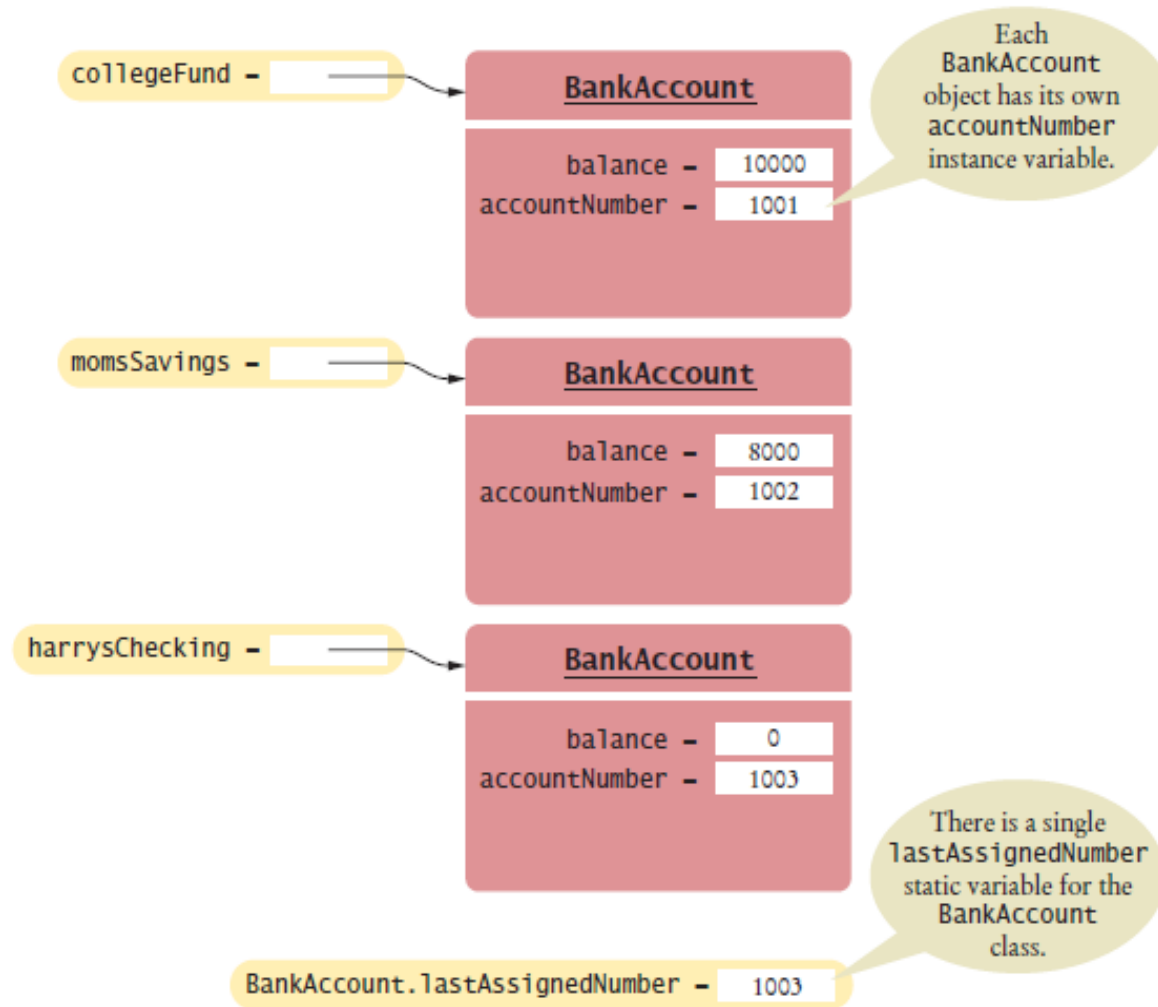
- Per assegnare numeri di conto bancario in sequenza
 - Abbiamo bisogno di un singolo valore di `lastAssignedNumber` che è una proprietà della classe, non un qualsiasi oggetto della classe.
 - Dichiararlo usando la parola riservata `static`

```
public class BankAccount
{
    private double
    balance;
    private int
    accountNumber;
    private static int lastAssignedNumber =
    1000;
    public BankAccount()
    {
        lastAssignedNumber++;
        accountNumber = lastAssignedNumber;
    }
    . . .
}
```

Variabili statiche e metodi statici

- Ogni oggetto BankAccount ha il proprio saldo e accountNumber come variabili di istanza
- Tutti gli oggetti condividono una singola copia della variabile lastAssignedNumber
- Tale variabile è archiviata in una posizione separata, al di fuori di qualsiasi oggetto BankAccount
- Le variabili statiche devono sempre essere dichiarate come private,
 - Ciò garantisce che i metodi di altre classi non modifichino i loro valori
- le costanti statiche possono essere private o pubbliche

Variabili statiche e metodi statici



Variabili statiche e metodi statici

- A volte una classe definisce metodi che non vengono invocati su un oggetto
 - Chiamati metodi statici
- Esempio: metodo `sqrt` della classe `Math`: se `x` è un numero, la chiamata `x.sqrt()` non è legale
- La classe `Math` fornisce un metodo statico: invocato come `Math.sqrt(x)`
- Nessun oggetto della classe `Math` è costruito.
- Il qualificatore `Math` dice semplicemente al compilatore dove trovare il metodo `sqrt`

Variabili statiche e metodi statici

Puoi definire i metodi statici:

```
public class Financial
{
    /**
     * Computes a percentage of an amount.
     * @param percentage the percentage to apply
     * @param amount the amount to which the
     * percentage is applied
     * @return the requested percentage of the
     * amount
     */
    public static double percentOf(double
percentage, double amount)
    {
        return (percentage / 100) * amount;
    }
}
```

Variabili statiche e metodi statici

Quando si chiama un tale metodo, bisogna fornire il nome della classe che lo contiene:

```
double tax = Financial.percentOf(taxRate,  
total);
```

Il metodo principale è sempre statico.

- All'avvio del programma, non ci sono oggetti.
- Pertanto, il primo metodo di un programma deve essere un metodo statico.

Pacchetti



Pacchetti

Pacchetto: insieme di classi correlate
Pacchetti importanti nella libreria Java:

Package	Compito	Esempio classe
java.lang	supporto linguaggio	Math
java.util	utilità	Random
java.io	input e output	PrintStream
java.awt	finestre toolkit	Color
java.net	networking	Socket
java.sql	gestione DB	ResultSet
javax.swing	interfaccia utente Swing	JButton
org.w3c.dom	modella documenti XML	Document

Pacchetti

Per inserire le classi in un pacchetto, devi inserire una riga

`package packageName;`

come prima istruzione nel file sorgente contenente le classi.

- Il nome del pacchetto è costituito da uno o più identificatori separati da punti.
- Per inserire la classe `Financial` in un pacchetto denominato `com.horstmann.bigjava`, il file `Financial.java` deve iniziare come segue:

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

Pacchetti

Un pacchetto speciale: pacchetto predefinito

- Non ha nome
- Nessuna dichiarazione sul pacchetto

Se non hai incluso alcuna istruzione del pacchetto nella parte superiore del tuo file di origine le tue classi sono inserite nel pacchetto predefinito.

Importare pacchetti

- Si può usare una classe senza importare: basta fare riferimento ad essa con il suo nome completo (nome del pacchetto più nome della classe):

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

Sconveniente

- La direttiva import ti consente di fare riferimento a una classe di un pacchetto con il suo nome di classe, senza il prefisso del pacchetto
- Per importare tutte le classi di un pacchetto:

```
import java.util.*;
```

Importare pacchetti

- Non è mai necessario importare `java.lang`.
- Non è necessario importare altre classi nello stesso pacchetto.
- Usa i pacchetti per evitare conflitti di nomi:

```
java.util.Timer
```

```
javax.swing.Timer
```

- I nomi dei pacchetti devono essere univoci.
- Per ottenere un nome di pacchetto: capovolgi il nome di dominio o l'email:
`com.horstmann.bigjava`

Importare pacchetti

Syntax `package packageName;`

The classes in this file
belong to this package.

`package com.horstmann.bigjava;`

A good choice for a package name
is a domain name in reverse.

Pacchetti e file di codice sorgente

- Il percorso di un file sorgente di classe deve corrispondere al nome del pacchetto a cui appartiene la classe.
- Le parti del nome tra i punti rappresentano le directory nidificate successivamente.
- Directory di base: contiene i file del tuo programma
- Posiziona la sottodirectory all'interno della directory di base.
- Se il tuo compito è in una directory
`/home/britney/hw8/problema1`
- Posizionare i file di classe nel pacchetto `com.horstmann.bigjava` nella directory:
 - `/home/britney/hw8/problem1/com/horstmann/bigjava` (UNIX)
 - Or `c:\Users\Britney\hw8\problem1\com\horstmann\ bigjava` (Windows)

Pacchetti e file di codice sorgente

- `/home/britney/hw8/problem1/com/horstmann/bigjava` (UNIX)
- **Or** `c:\Users\Britney\ hw8\problem1\com\horstmann\bigjava`
(Windows)

