04
Uso avanzato di Eclipse, ereditarietà e classi astratte

Danilo Pianini
Giovanni Ciatto, Angelo Croatti, Mirko Viroli

C.D.L. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

13 ottobre 2017

Outline

Goal della lezione

- Utilizzo avanzato di Eclipse
  - Debugging
  - Refactoring
  - Keyboard Shortcuts
- Imparare a costruire classi sfruttando l'ereditarietà
  - Scrittura ed estensione di (1) classi e (2) classi astratte
- Ragionamenti sul (buon) design di Software OOP-oriented

Outline

Outline

Debugging di Applicazioni [1, 2] I

Motivazioni

- Difficilmente le applicazioni software risultano essere totalmente esenti da problemi/errori alla prima stesura del codice sorgente
- Spesso lo sviluppatore tende a sottovalutare e/o ignora alcuni effetti collaterali (*side-effects*) che porzioni di codice sorgente possono provocare
- In genere, un software privo di errori lo si ottiene step-by-step
  - Una buona progettazione a priori consente di ridurre il numero di bug che si potrebbero inserire nel codice sorgente, tuttavia...
  - ...qualche bug sarà inevitabilmente presente, e dovrà essere identificato e corretto!

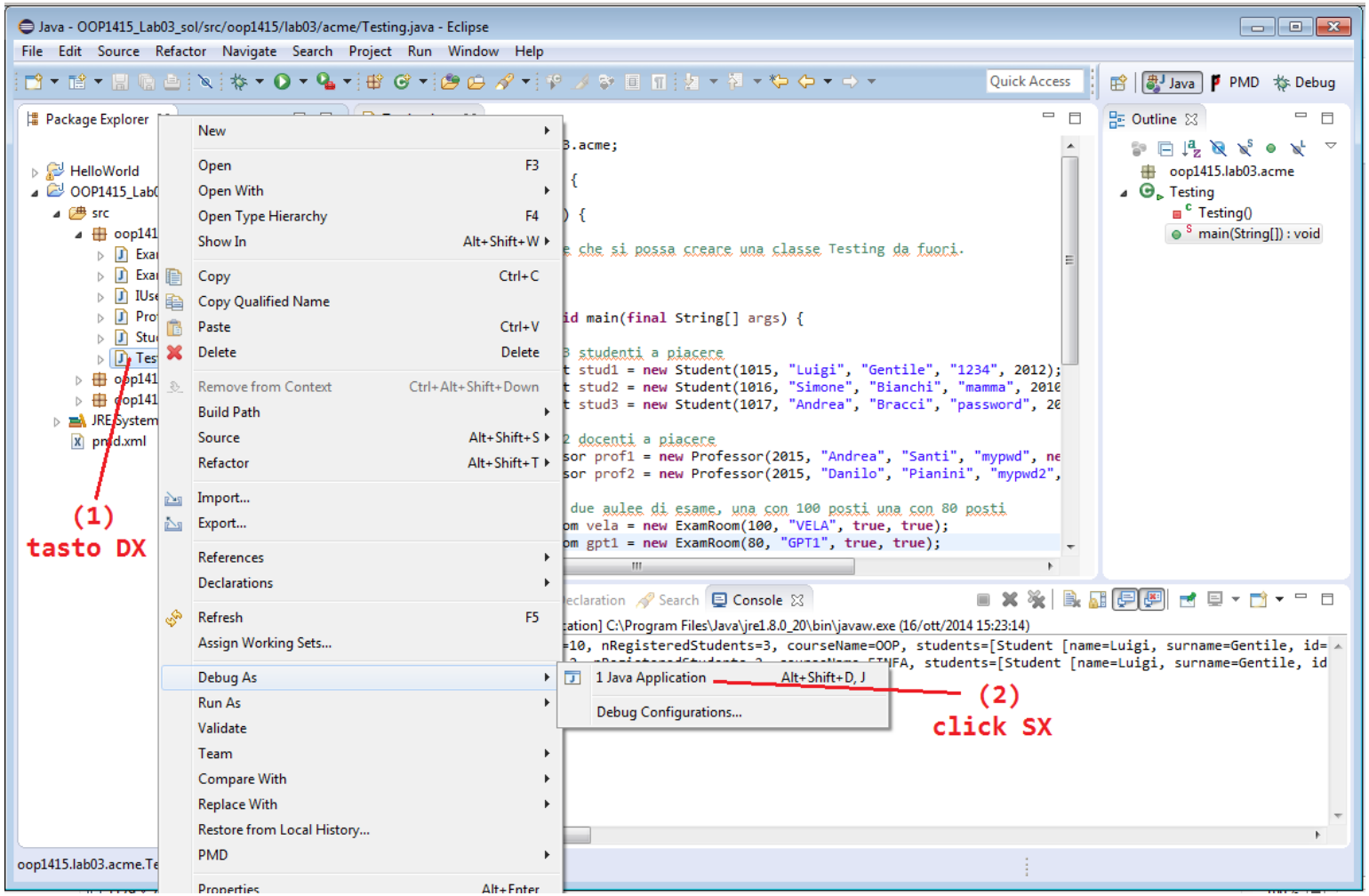
Debugging di Applicazioni [1, 2] II

Approcci

1. Meccanismi di logging
   - Si aggiunge al codice sorgente la possibilità di fare logging in modo da tracciare lo stato interno del programma in fase di esecuzione: valore corrente di variabili, campi, parametri, ...
   - Utile in alcuni casi, ma in generale **da evitare**!
   - Appesantisce il sorgente
   - Appesantisce il software (a meno di non usare opportune librerie)
   - Basta un side effect per generare un Heisenbug [3]
2. Utilizzo di strumenti di debugging forniti dall'IDE
   - Controllo efficace sull'esecuzione dell'applicazione
   - Ispezionabilità a run-time
   - Potenti strumenti per indagare cosa succede nel programma
   - Molto difficile che causi Heisenbugs
     - Finché non si mette in mezzo la concorrenza...

Debugging di Applicazioni in Eclipse

- L'avvio del debugging è molto simile all'esecuzione delle applicazioni
  - (menu contestuale di progetto) > Debug As > Java Application



Debugging in Eclipse: manipolazione del flusso di controllo

- L'avvio del debugger in Eclipse comporta l'apertura della perspective *Debug*, con gli strumenti utili per le operazioni di debugging

Principali operazioni di manipolazione del flusso di controllo

1. Inserimento di un breakpoint
   - Doppio click sulla linea di interesse (o CTRL+SHIFT+B)
2. Esecuzione step-by-step: F6
   - Una volta che l'esecuzione è stata sospesa da un breakpoint
3. Step into: F5
   - Debugging corpo di un metodo/costruttore
4. Step return: F7
   - Ritorno dal debugging del corpo di un metodo/costruttore
5. Resume: F8
   - Esecuzione fino al prossimo breakpoint

Debugging in Eclipse: strumenti I

Breakpoint condizionali

Tramite click destro su breakpoint e "Properties", è possibile scrivere una condizione per il breakpoint, ad esempio di scattare solo dopo un certo numero di volte, oppure di scattare se una certa condizione è vera (estremamente utile).

Osservazione e manipolazione dei valori

La view "Variables" consente di visualizzare il valore di tutti i campi e delle variabili locali, esplorando anche l'interno degli oggetti. Inoltre, consente di modificarne i valori a runtime, per ispezionare il funzionamento del programma.

Debugging in Eclipse: strumenti II

Valutazione di espressioni

È possibile, nella tab "Expressions", scrivere un'espressione java valida nel contesto (quindi, con accesso alle variabili locali e ai campi). L'IDE la valuterà e scriverà il risultato. Molto utile per visualizzare valori che richiederebbero altrimenti l'inserimento di una stampa ed il riavvio del debug, oppure un uso complicato dell'osservazione dentro Variables.

Debugging in Eclipse: strumenti III

Iniezione di codice a caldo

Nel caso in cui si effettui una modifica al software mentre è in esecuzione in modalità debug e lo si ricompili (o si abbia la modalità di autocompilazione attivata), Eclipse tenterà di "iniettare" i nuovi compilati all'interno della JVM in esecuzione:

- Viene "scaricata" la classe precedentemente inserita nel classpath
- Viene "iniettata" la nuova classe

L'operazione può fallire, nel qual caso l'IDE notificherà che ci sono in esecuzione classi e metodi obsoleti. Per massimizzare la riuscita:

1. Scegliere uno stack frame precedente all'uso della classe da sostituire;
2. Usare la funzione "Drop to frame" per riportare il flusso di controllo all'inizio dello stack frame che si sta eseguendo;
   - Eventuali side effects non vengono cancellati!
   - (altro esempio in cui l'immutabilità è d'aiuto)
3. Solo a questo punto salvare il sorgente modificato.

Debugging: Test delle Principali Operazioni

Classe d'esempio: `it.unibo.oop.lab04.bank.DebugTestBankAccount`

```java
package it.unibo.oop.lab04.bank;

public class DebugTestBankAccount {

    private DebugTestBankAccount() {}

    public static void main(final String[] args) {
        final AccountHolder usr1 = new AccountHolder("Mario", "Rossi", 1);
        final AccountHolder arsenioLupin = new AccountHolder("Arsenio", "Lupin", 2);

        final BankAccount b1 = new SimpleBankAccount(usr1.getUserID(), 1000);

        b1.deposit(usr1.getUserID(), 500);
        b1.deposit(usr1.getUserID(), 1000);
        b1.withdraw(1, 700);
        b1.depositFromATM(usr1.getUserID(), 100);

        System.out.println("Mario Rossi current balance is " + b1.getBalance());

        for (int i = 0; i < 5; i++) {
            b1.withdraw(1, 500);
        }

        b1.deposit(usr1.getUserID(), 2000);
        System.out.println("Mario Rossi current balance is " + b1.getBalance());

        b1.withdraw(arsenioLupin.getUserID(), 3000);
        System.out.println("Mario Rossi current balance is " + b1.getBalance());
    }
}
```

Outline

Refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.
— `refactoring.com`

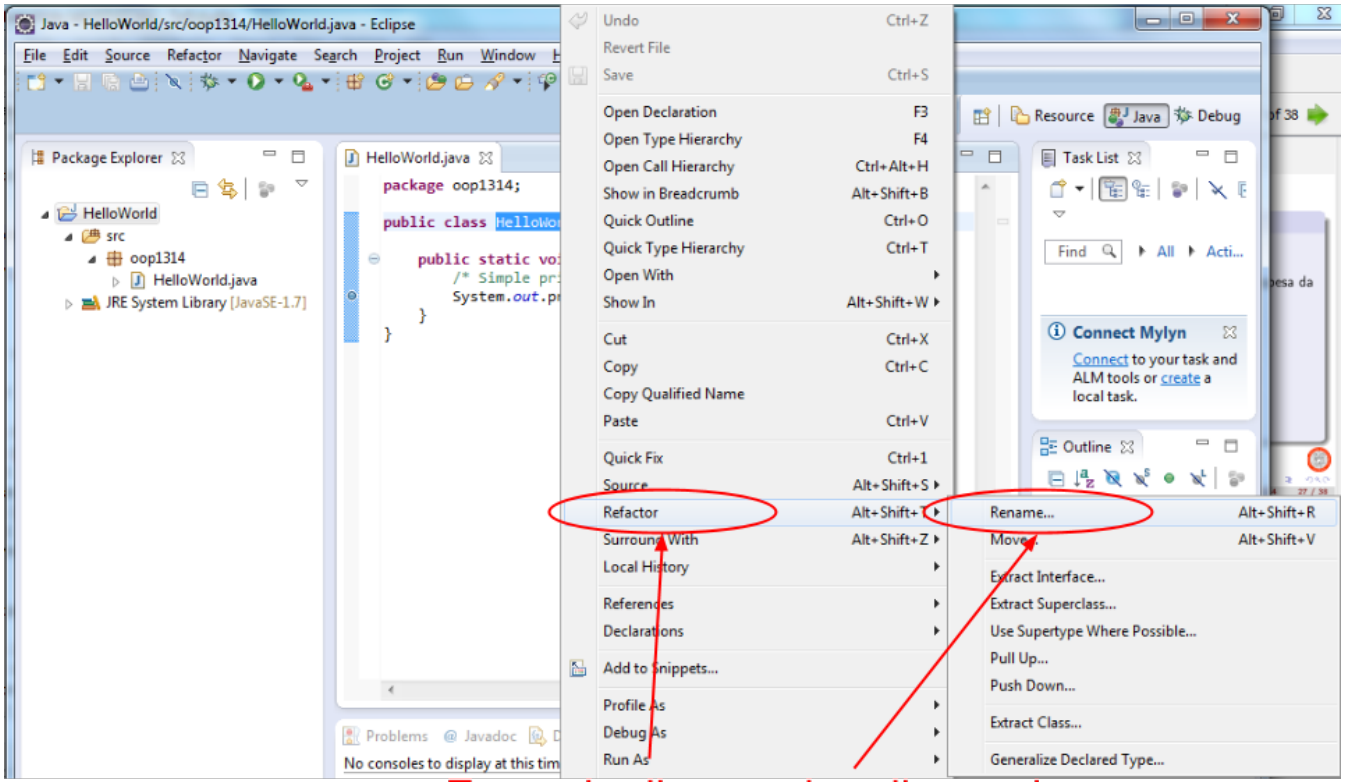
Vantaggi nel refactoring supportato dall'IDE

- Le modifiche sono gestite dall'IDE in maniera consistente
  - Es (1). *Rinominare una variabile*: se si procede "a mano", si dovrà modificare il nome della variabile sia nella riga in cui la si dichiara sia in tutte le sue occorrenze di utilizzo. Viceversa, avvalendosi dell'IDE, la modifica da fare è una sola.
  - Es (2). *Spostare una classe da un package ad un altro*: utilizzando l'IDE vengono aggiornati tutti i riferimenti nell'intero progetto
- Minimizza l'introduzione di errori in fase di refactoring

Refactoring in Eclipse

Operazioni di modifica del nome di variabili, classi, metodi etc.

- Gestite in maniera automatica e safe dall'IDE
- Attivabile anche tramite ALT+SHIFT+R sulla selezione di interesse



Esempio di renaming di una classe

Outline

Keyboard Shortcuts 1/2

Perchè usarle?

- Velocizzano l'esecuzione di operazioni ricorrenti
- Diminuiscono la probabilità di *"infortuni"* legati all'uso del mouse
  - I tennisti hanno problemi al gomito, gli informatici al tunnel carpale
  - (più terminale $\Rightarrow$ più salute)

Principali shortcut (per Mac OSX sostiuire CMD a CTRL)

- CTRL+1: quick-fix contestuale per errori (molto potente)
- ALT+SHIFT+R: refactoring di campi/metodi/classi
- CTRL+SHIFT+/: commento delle linee selezionate
- CTRL+PGUP/PGDOWN: per muoversi di uno step avanti (PGUP) o indietro (PGDOWN) tra la lista di sorgenti correntemente aperti
- F3 (o CTRL+CLICK): ci sposta alla definizione di un dato elemento
- CTRL+SHIFT+L: lista delle shortcut disponibili per il dato contesto

Keyboard Shortcuts 2/2

Altre shortcut (per Mac OSX sostiuire CMD a CTRL)

- CTRL+SHIFT+O: include in automatico tutti gli import necessari sulla base delle classi utilizzate nel sorgente corrente
- CTRL + .: sposta il cursorse al successivo errore/warning
- CTRL+F8: consente di spostarsi tra le varie perspective
- CTRL+J: search incrementale, senza l'uso di GUI
- ALT+SHIFT+S: da l'accesso a un insieme di wizard con cui automatizzare la scrittura di costruttori, getter, setter, etc.

Outline

Preparazione Ambiente di Lavoro 1/2

- Accendere il PC
- Loggarsi sul sito del corso
  - https://bit.ly/oop2016cesena
- Scaricare dalla sezione `lab` del sito il file `lab04.zip` contenente il materiale dell'esercitazione odierna
- Spostare il file scaricato sul Desktop
- Decomprimere il file usando 7zip (o un programma analogo) sul Desktop
- Copiare la cartella scompattata (`lab04`) nel vostro workspace di Eclipse
  - p.e. `C:\Users\<username>\workspace`
- Importare il progetto `lab04` con la procedura di importazione dei progetti vista durante lo scorso laboratorio

Modalità di Lavoro

1. Leggere la consegna
2. Risolvere l'esercizio in autonomia
3. Cercare di risolvere autonomamente eventuali piccoli problemi che possono verificarsi durante lo svolgimento degli esercizi
4. Utilizzare le funzioni di test presenti nei sorgenti per il testing dell'esercizio
5. Contattare i docenti nel caso vi troviate a lungo bloccati nella risoluzione di uno specifico esercizio
6. A esercizio ultimato contattare i docenti per un rapido controllo della soluzione realizzata
7. Proseguire con l'esercizio seguente

Outline

Outline

SimpleBankAccount e StrictBankAccount

Miglioriamo la precedente esercitazione

- Nelle soluzioni dell'esercizio della volta scorsa, c'è molto codice duplicato fra `SimpleBankAccount` e `StrictBankAccount`
- Si crei la classe `ExtendedStrictBankAccount extends SimpleBankAccount`, con lo stesso comportamento di StrictBankAccount, cercando di ottimizzare il riuso di codice.
- È **proibito** apportare modifiche a `SimpleBankAccount`, di qualunque tipo.
- Si mostri il risultato al docente per una correzione

Una nuova architettura

- La soluzione precedente è stata ottenuta per modifica di un design sub-ottimo (non conoscevamo ancora l'ereditarietà!)
- Come sarebbe stato il design dell'applicazione se avessimo conosciuto fin dall'inizio il meccanismo di ereditarietà?

SimpleBankAccount e StrictBankAccount

Una nuova architettura

- All'interno del package `it.unibo.oop.lab04.bank2`, si creino le seguenti classi:
  - `abstract AbstractBankAccount implements BankAccount`
    - dovrà contenere elementi comuni a **tutti** i conti correnti
    - **tutti** i campi della classe dovranno essere privati
    - la classe dovrà esporre come metodi `public` tutti e soli i metodi dell'interfaccia `BankAccount`
    - deve definire ed utilizzare due metodi `protected abstract: boolean isWithdrawAllowed(double)` (`true` se se è possibile prelevare dal conto il valore passato) e `double computeFee()` (ritorna l'ammontare complessivo dei costi di gestione)
  - `ClassicBankAccount extends AbstractBankAccount`
    - Implementa lo stesso comportamento di `SimpleBankAccount`
  - `RestrictedBankAccount extends AbstractBankAccount`
    - Implementa lo stesso comportamento di `StrictBankAccount`
- In cosa è migliore della soluzione precedente?

Outline

Costruzione e Testing di Robot

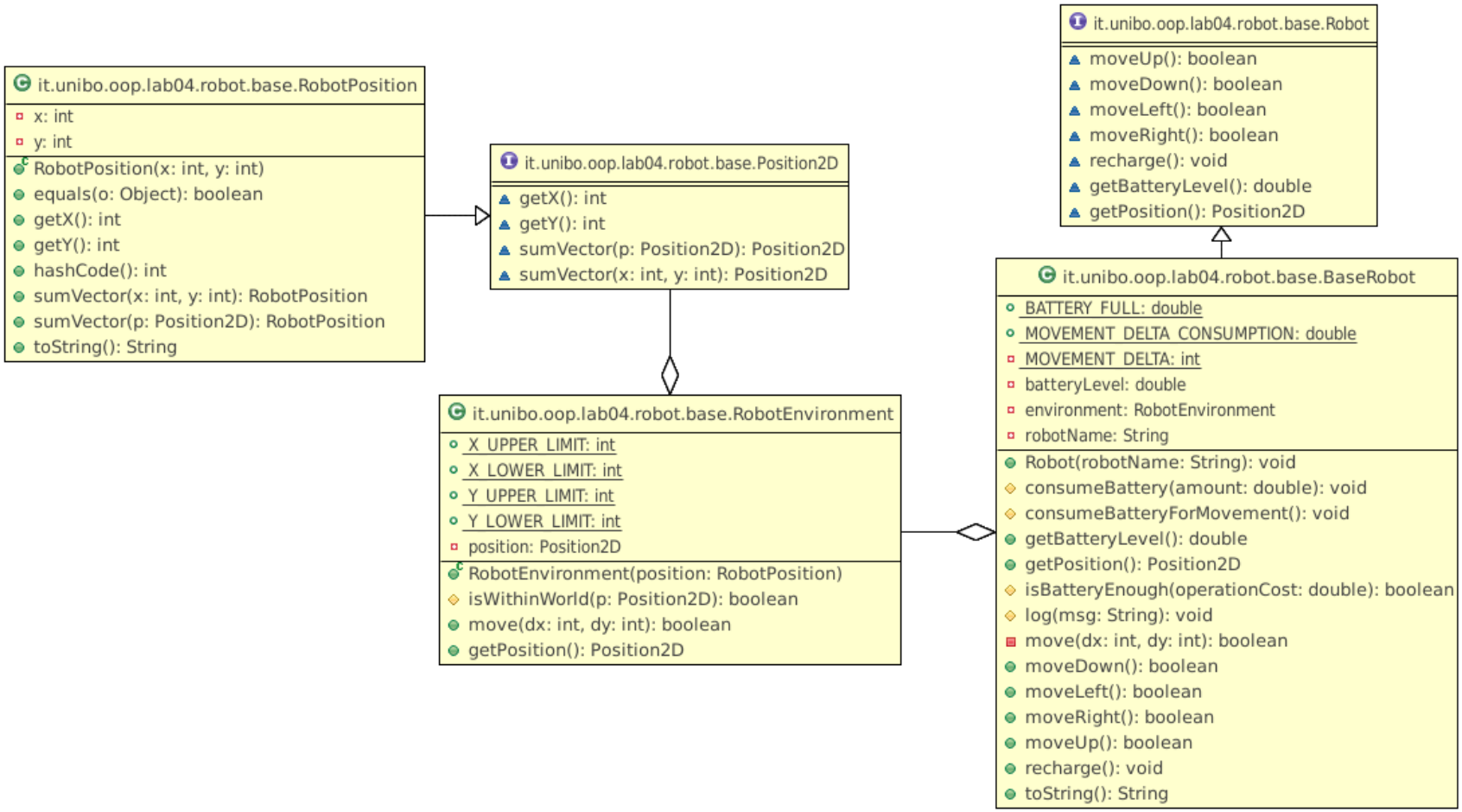
Nella lezione di oggi studieremo in pratica l'ereditarietà e le classi astratte usando come scenario di riferimento la modellazione e il testing di robot

- `package it.unibo.oop.lab04.robot.base`: set di interfacce e classi fornite

- `Robot`
- `Position2D`
- `BaseRobot`
- `RobotPosition`
- `RobotEnvironment`

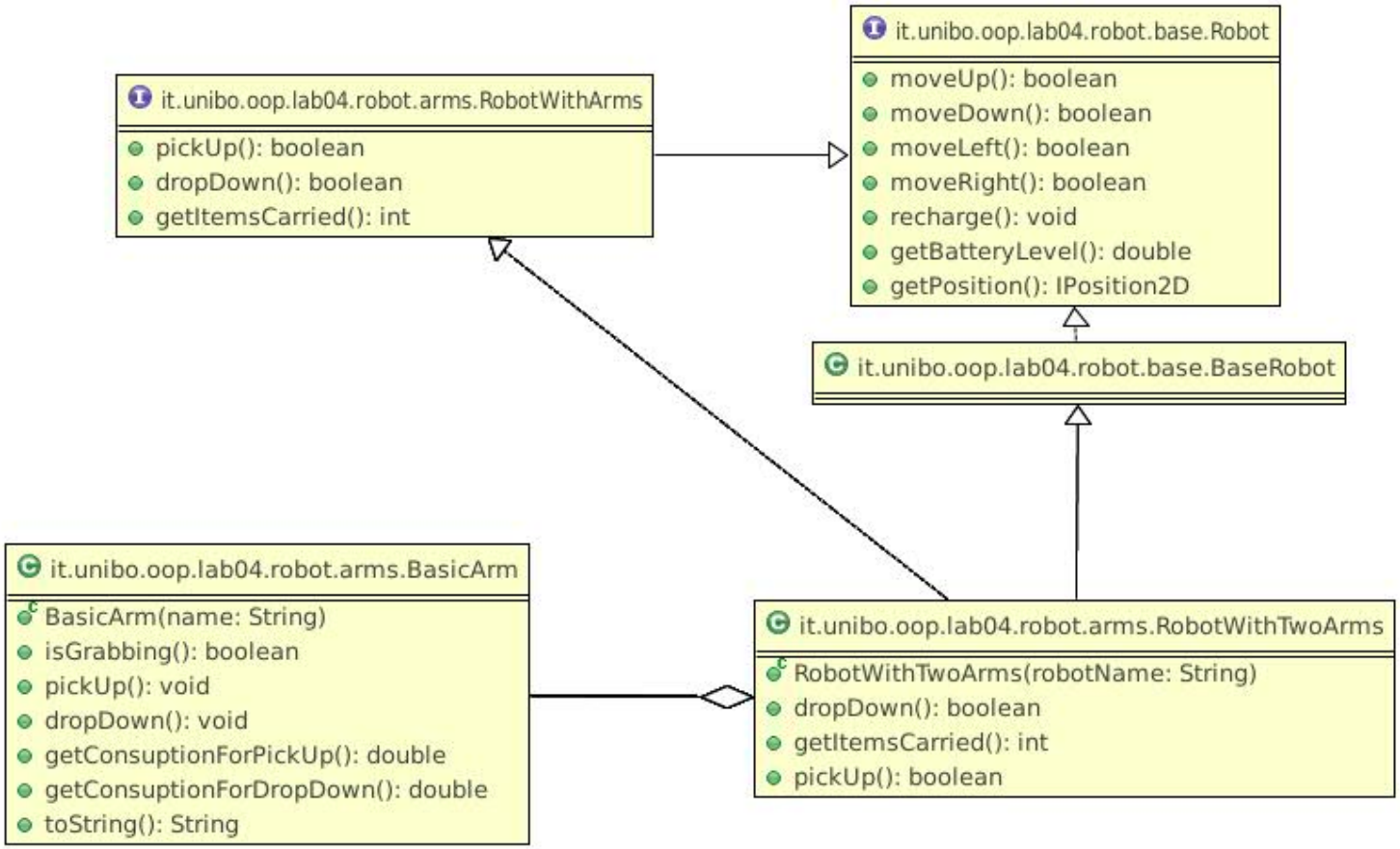
Si utilizzi `TestRobots` per capire il funzionamento delle classi suddette

Costruzione e Testing di Robot



Outline

Suggested design



Design guidato

Sfruttando le classi base già definite, e facendo riferimento allo schema UML allegato nella slide precedente, si realizzino nel `package it.unibo.oop.lab04.robot.arms`:

- `interface RobotWithArms`: descrive un robot dotato di braccia. Presenta tre metodi:
  - `boolean pickUp()` raccoglie un oggetto, restituisce `true` se ha successo
  - `boolean dropDown()` rilascia un oggetto, restituisce `true` se ha successo
  - `int getItemsCarried()` restituisce il numero di oggetti che si stanno trasportando.
- `class BasicArm`: modella un singolo braccio robot. Si faccia riferimento allo schema UML precedente, sapendo che ciascun braccio è in grado di sollevare un solo oggetto alla volta, che ha un consumo per prendere un oggetto e per lasciarlo.
- `class RobotWithTwoArms extends BaseRobot implements RobotWithArms`: modella un robot con due braccia. Al comando `pickUp()`, il robot preleva un oggetto, aumentando il numero di oggetti che sta trasportando, ed occupando una delle braccia. Quando tutte le braccia sono occupate, `pickUp()` fallisce (restituisce `false`). Allo stesso modo, quando le braccia sono vuote, `dropDown()` fallisce. Quando il robot sta trasportando oggetti, il suo consumo di batteria è superiore.

Si utilizzi `TestRobots` per capire il funzionamento delle classi suddette

Outline

Design di una applicazione

- Come ingegneri, dovrete essere in grado di tradurre una descrizione in linguaggio naturale ("a parole") di una applicazione in termini di astrazioni adatte ad essere scritte in forma di software (nel nostro caso, in elementi di linguaggio di programmazione orientato agli oggetti)
- Si tratta dell'esercizio **più difficile** e **più importante** che dovrete fare quando realizzerete il progetto
- Dalle vostre capacità di analisi del problema e di design della soluzione dipende il successo o meno del vostro software
- Nel prossimo esercizio, vi sarà data una descrizione in linguaggio naturale, e starà a voi realizzare il software autonomamente
- **SUGGERIMENTO** — Prima di iniziare a "sporcarsi le mani" col codice, prendete carta e penna e cercate di ottenere un design chiaro dei componenti che realizzerete: nessuno costruisce un'automobile partendo con l'assemblare i bulloni, ma decide prima quali saranno i componenti e come andranno connessi fra loro.

Outline

Requisiti: robot componibile

- Si desidera realizzare un robot di tipo componibile.
- Un robot componibile è un robot al quale possono essere aggiunti o rimossi nuovi componenti arbitrari.
- Il robot componibile espone una funzionalità che consente di mettere in moto e far funzionare tutti i componenti connessi, a patto ovviamente che siano accesi.
- Quando tale funzionalità viene chiamata, il robot mette in funzione in ordine, per una sola volta, tutti i componenti ad esso connessi.

Requisiti: componenti di robot

- Un componente per robot è un'entità che può essere accesa o spenta.
- Il componente può essere non connesso oppure connesso ad un solo robot.
- Il componente è in grado di compiere operazioni sul robot.
- Ciascun componente ha un proprio consumo di energia.
- Quando il robot a componenti fa uso del componente, deve scalare il consumo di energia del componente dalla propria batteria.
- Alcuni componenti sono in grado di supportare dei comandi.
- Ciascun componente comandabile ha il proprio set di comandi
- Il componente comandabile può ricevere un comando
- Alla ricezione del comando, se il comando corrisponde ad uno dei comandi supportati, il componente comandabile cambia il proprio comportamento in maniera tale da eseguire il comando richiesto.

Requisiti: test

Si desidera testare l'infrastruttura creando un robot componibile ed assegnandogli i seguenti componenti:

- Batteria atomica:
  - Componente non comandabile.
  - Batteria alimentata ad Uranio-239, che ricarica istantaneamente il robot.
  - In fase di test, per evitare il surriscaldamento, si attivi la batteria atomica solo nel caso in cui la batteria del robot sia al di sotto del 50%.
- Navigatore di confine:
  - Componente non comandabile.
  - Una volta avviato, fa sì che il robot raggiunga il bordo del `RobotEnvironment` e continui ad esplorarlo.
- Due braccia prensili:
  - Componente comandabile, con comandi "pick" e "drop".
  - Se è attivo il comando "pick" e il braccio non ha alcun oggetto in mano, allora ne viene preso uno.
  - Se è attivo il comando "pick" e il braccio ha già un oggetto in mano, allora non viene fatta alcuna azione.
  - Se è attivo il comando "drop" e il braccio ha già un oggetto in mano, allora l'oggetto viene lasciato.
  - Se è attivo il comando "drop" e il braccio non ha alcun alcun oggetto in mano, allora non viene fatta alcuna azione.

Bibliography I

[1] Wikipedia.

Debugger.

Online, available at: http://en.wikipedia.org/wiki/Debugger – Last Retrieved: October 14, 2016.

[2] Wikipedia.

Debugging.

Online, available at: http://en.wikipedia.org/wiki/Debugging – Last Retrieved: October 14, 2016.

[3] Wikipedia.

Heisenbug.

Online, available at: https://en.wikipedia.org/wiki/Heisenbug – Last Retrieved: October 14, 2016.