Programmazione in Java
e
gestione della grafica

Lezione 8

La scorsa lezione…..

- Metodo **`Math.random()`**
- Esempi
- Strategie di ricerca binaria: il gioco “Indovina il numero”

Gioco “Indovina il numero”: utente vs computer

ALGORITMO

- Il computer “chiede” all’utente di pensare un numero x tra 1 e 100 e chiede di inserire 0 per iniziare il gioco
- Il computer “propone” un n… …tarlo (0= ok, 1= troppo piccolo, 2=
- Finchè l’utente non scrive 0,
  - **“propone” un nuovo numero**
  - Chiede nuovamente all’utente di valutarlo
  - (mantiene memoria del numero dei tentativi)
- Se *entro* **6** tentativi viene inserito y corretto (y=x)
  - il computer “si vanta” con l’utente della sua vittoria riportando il numero di tentativi fatti per indovinare il numero

  altrimenti
  - Il computer comunica all’utente che di essere stato battuto e termina il gioco

Occore implementare una strategia efficiente di gioco!

Esercizio della lezione scorsa

- Esercizio 7.0

  Descrivere nei particolari un algoritmo per implementare una strategia efficace per giocare a IndovinaNumero

Strategia per IndovinaNumero

ALGORITMO IndovinaNum

Sia [min, max] l'intervallo in cui si trova il numero **goal** da indovinare

min=1 , max=100

Scelgo num punto medio dell'intervallo:

num = (max –min)/2 +min

Finchè (num != goal)

Se num > goal ("troppo grande")

max =goal (vado nel'intervallo [min, goal]

Se num < goal ("troppo piccolo")

min =goal (vado nel'intervallo [goal,max]

num = (max –min)/2 +min

Dimostrazione correttezza

Proposizione: L'algoritmo IndovinaNum **termina** sempre e individua il numero goal in **<= $\log_2$ 100** tentativi.

Dim:

Ad ogni iterazione l'intervallo contiene goal e ha dimensioni metà della volta precedente.

Dopo **$\log_2$ 100** iterazioni l'intervallo contiene 1 solo numero (che è proprio goal).

Si esce dal ciclo in **$\log_2$ 100** passi.

Nota: Il numero di volte in cui per arrivare a 1 partendo da 100 devo dividere a metà è proprio **$\log_2 100$**

Sul compilatore :

`IndovinaNumero2.java`

Esercizi

- Esercizio 8.1

  Modificare IndovinaNumero2 inserendo i controlli sull'input.

  (sia che venga inserito effettivamente un numero come richiesto, sia che l'inserimento della successione di 1 e 2 corrispondenti a "troppo piccolo" e "troppo grande" sia coerente).

Esercizio della lezione scorsa

- Esercizio 7.2

  Un numero n>1 si dice *primo* se non ammette altri divisori oltre 1 e se stesso.

  Scrivere un metodo **`Primo`** che prende come parametro un numero intero positivo e restituisce un valore booleano true se tale numero e' primo e false altrimenti.

Esercizio della lezione scorsa

- Esercizio 7.3

  Scrivere un programma (che utilizza il metodo **`Primo`**) che prende in input dall'utente un intero positivo e dice se tale numero e' o no primo.

  (L'utente potrebbe essere invitato ad inserire sempre nuovi numeri finche' non inserisce lo 0)

Esercizio della lezione scorsa

- Esercizio 7.3

  Scrivere un programma (che utilizza il metodo **`Primo`**) che scrive sullo schermo tutti i numeri primi <=1000.

ALGORITMO Primo

num : numero a cui applico il test

d :eventuale divisore

maxd: massimo valore di d da testare

vprimo : (variabile booleana)


```
vprimo= true
d=2
Finchè ( d<= maxd AND vprimo)
    if ( num % d ==0)
        vprimo = false
    d++
```

Sul compilatore :

`TestPrimo.java`

INIZIAMO A PARLARE DI CLASSI E OGGETTI

(si prosegue in inglese)

3.2 Classes, Objects, Methods and Instance Variables

- Class provides one or more methods
- Method represents task in a program
  - Describes the mechanisms that actually perform its tasks
  - Hides from its user the complex tasks that it performs
  - Method call tells method to perform its task

3.2 Classes, Objects, Methods and Instance Variables (Cont.)

- Classes contain one or more attributes
  - Specified by instance variables
  - Carried with the object as it is used

3.3 Declaring a Class with a Method and Instantiating an Object of a Class

- Each class declaration that begins with keyword `public` must be stored in a file that has the same name as the class and ends with the `.java` file-name extension.

Class `GradeBook`

- keyword `public` is an access modifier
- Class declarations include:
  - Access modifier
  - Keyword `class`
  - Pair of left and right braces

Class `GradeBook`

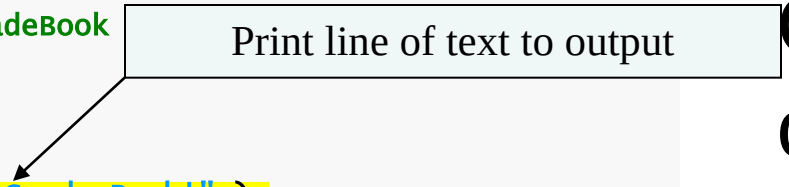
- Method declarations
  - Keyword `public` indicates method is available to public
  - Keyword `void` indicates no return type
  - Access modifier, return type, name of method and parentheses comprise method header

- Declaring more than one `public` class in the same file is a compilation error.

Outline

GradeBook.java

```
1  // Fig. 3.1: GradeBook.java
2  // Class declaration with one method.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11
12 } // end class GradeBook
```



Class `GradeBookTest`

- Java is extensible
  - Programmers can create new classes
- Class instance creation expression
  - Keyword `new`
  - Then name of class to create and parentheses
- Calling a method
  - Object name, then dot separator (`.`)
  - Then method name and parentheses

Outline

- GradeBookTest.java

```
1  // Fig. 3.2: GradeBookTest.java
2  // Create a GradeBook object and call its displayMessage method.
3
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9        // create a GradeBook object and assign it
10       GradeBook myGradeBook = new GradeBook();
11
12       // call myGradeBook's displayMessage metho
13       myGradeBook.displayMessage();
14    } // end main
15
16 } // end class GradeBookTest
```

Use class instance creation expression to create object of class `GradeBook`

Call method `displayMessage` using `GradeBook` object

```
Welcome to the Grade Book!
```

Compiling an Application with Multiple Classes

- Compiling multiple classes
  - List each `.java` file in the compilation command and separate them with spaces
  - Compile with `*.java` to compile all `.java` files in that directory

3.4 Declaring a Method with a Parameter

- Method parameters
  - Additional information passed to a method
  - Supplied in the method call with arguments

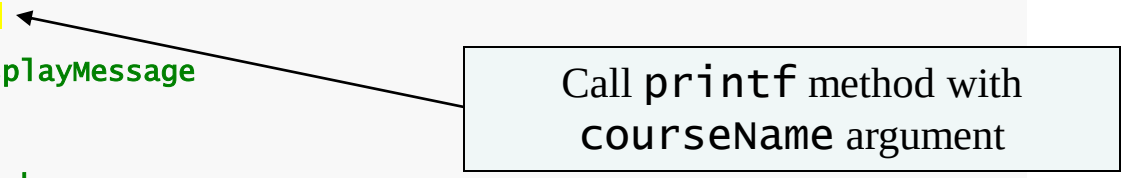
3.4 Declaring a Method with a Parameter

- `Scanner` methods
  - `nextLine` reads next line of input
  - `next` reads next word of input

Outline

- GradeBook.java

```
1  // Fig. 3.4: GradeBook.java
2  // Class declaration with a method that has a parameter.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9        System.out.printf( "Welcome to the grade book for\n%s!\n",
10          courseName );
11    } // end method displayMessage
12
13 } // end class GradeBook
```



Outline

- GradeBookTest.java

```
1  // Fig. 3.5: GradeBookTest.java
2  // Create GradeBook object and pass a String to
3  // its displayMessage method.
4  import java.util.Scanner; // program uses Scanner
5
6  public class GradeBookTest
7  {
8     // main method begins program execution
9     public static void main( String args[] )
10    {
11       // create Scanner to obtain input from command window
12       Scanner input = new Scanner( System.in );
13
14       // create a GradeBook object and assign it
15       GradeBook myGradeBook = new GradeBook();
16
17       // prompt for and input course name
18       System.out.println( "Please enter the course name:" );
19       String nameOfCourse = input.nextLine(); //
20       System.out.println(); // outputs a blank l
21
22       // call myGradeBook's displayMessage method
23       // and pass nameOfCourse as an argument
24       myGradeBook.displayMessage( nameOfCourse );
25    } // end main
26
27 } // end class GradeBookTest
```

Call `nextLine` method to read a line of input

Call `displayMessage` with an argument

```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Software Engineering Observation 3.1

- Normally, objects are created with `new`. One exception is a string literal that is contained in quotes, such as `"hello"`. String literals are references to `String` objects that are implicitly created by Java.

More on Arguments and Parameters

- Parameters specified in method's parameter list
  - Part of method header
  - Uses a comma-separated list

Notes on `Import` Declarations

- `java.lang` is implicitly imported into every program
- Default package
  - Contains classes compiled in the same directory
  - Implicitly imported into source code of other files in directory
- Imports unnecessary if fully-qualified names are used

3.5 Instance Variables, *set* Methods and *get* Methods

- Variables declared in the body of method
  - Called local variables
  - Can only be used within that method
- Variables declared in a class declaration
  - Called fields or instance variables
  - Each object of the class has a separate instance of the variable

Outline

- GradeBook.java

```
1  // Fig. 3.7: GradeBook.java
2  // GradeBook class that contains a courseName instance variable
3  // and methods to set and get its value.
4
5  public class GradeBook
6  {
7     private String courseName; // course name for this GradeBook
8
9     // method to set the course name
10    public void setCourseName( String name )
11    {
12       courseName = name; // store the course name
13    } // end method setCourseName
14
15    // method to retrieve the course name
16    public String getCourseName()
17    {
18       return courseName;
19    } // end method getCourseName
20
21    // display a welcome message to the GradeBook user
22    public void displayMessage()
23    {
24       // this statement calls getCourseName to get the
25       // name of the course this GradeBook represents
26       System.out.printf( "Welcome to the grade book for\n%s!\n",
27          getCourseName() );
28    } // end method displayMessage
29
30 } // end class GradeBook
```

Instance variable `courseName`

set method for `courseName`

get method for `courseName`

Call *get* method

`GradeBookTest` Class That Demonstrates Class `GradeBook`

- Default initial value
  - Provided for all fields not initialized
  - Equal to `null` for `Strings`

set and *get* methods

- `private` instance variables
  - Cannot be accessed directly by clients of the object
  - Use *set* methods to alter the value
  - Use *get* methods to retrieve the value

Outline

- GradeBookTest.java
- (1 of 2)

```
1  // Fig. 3.8: GradeBookTest.java
2  // Create and manipulate a GradeBook object.
3  import java.util.Scanner; // program uses Scanner
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       // create a GradeBook object and assign it to myGradeBook
14       GradeBook myGradeBook = new GradeBook();
15
16       // display initial value of courseName
17       System.out.printf( "Initial course name is: %s\n\n",
18          myGradeBook.getCourseName() );
19
```

Call *get* method for `courseName`

Outline

- GradeBookTest.java
- (2 of 2)

```
20       // prompt for and read course name
21       System.out.println( "Please enter the course name:" );
22       String theName = input.nextLine(); // read a line of text
23       myGradeBook.setCourseName( theName ); // s
24       System.out.println(); // outputs a blank line
25
26       // display welcome message after specifying course name
27       myGradeBook.displayMessage();
28    } // end main
29
30 } // end class GradeBookTest
```

Call *set* method for `courseName`

Call `displayMessage`

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Primitive Types vs. Reference Types

- Types in Java
  - Primitive
    - `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
  - Reference (sometimes called nonprimitive types)
    - Objects
    - Default value of `null`
    - Used to invoke an object's methods

Esercizi

- Esercizio 8.1

  Modificare IndovinaNumero2 inserendo i controlli sull'input.

  (controllare sia che venga inserito effettivamente un numero come richiesto, sia che l'inserimento della successione di 1 e 2 corrispondenti a "troppo piccolo" e "troppo grande" sia coerente).

Esercizi

- Esercizio 8.2

  Un numero n>1 si dice *perfetto* se la somma dei suoi divisori propri (cioe' <n) e' uguale ad n. (Esempio 6=3+2+1 e' perfetto).

  Scrivere un metodo che prende come parametro un numero intero positivo e stampa sullo schermo se e' perfetto oppure no visualizzando i suoi divisori propri.

  Utilizzare il metodo in un programma che prende in input dall'utente un intero positivo e dice se tale numero e' o no perfetto.

  (L'utente potrebbe essere invitato ad inserire sempre nuovi numeri finche' non inserisce lo 0)

Esercizi

- Esercizio 8.3

  Utilizzare il metodo Perfetto dell'esercizio precedente in un programma che prende in input dall'utente un intero positivo max e scrive sullo schermo tutti i numeri perfetti compresi tra 6 e max .