Esercitazione n° 2

Obiettivi

- Progettazione di una applicazione Java
  - Utilizzo di classi come “schemi”
  - Utilizzo di costruttori
  - Utilizzo di stringhe
  - Uguaglianza tra oggetti
  - Utilizzo di classi come componenti software statici
- Utilizzo di **Gel** come Editor

  Per la compilazione ed esecuzione usare ancora gli strumenti JDK…

Specifica del Problema

Si progetti un gioco che simuli il combattimento tra entità **Marziane** e **Terrestri**. Il comportamento astratto di tali entità è specificato nelle slide seguenti. Il componente software che gestisce il gioco dovrà creare un oggetto di tipo '*Marziano*' con matricola non definita ed uno invece con matricola "*x102*". Dovrà creare inoltre un oggetto di tipo '*Terrestre*' dal nome sconosciuto ed uno di nome "*Ken*". L'oggetto "*x102*" deve attaccare i Terrestri mentre "*Ken*" deve attaccare i Marziani. A seconda che vincano o perdano l'indice di vitalità di "*x102*" e "*Ken*" può salire o scendere. Inoltre, il componente software che gestisce il gioco dovrà farli entrambi attaccare due volte stampando in uscita il loro stato al termine di ogni attacco.

L'entità MARZIANO (1)

Ogni marziano ha un numero di **matricola**. Il suo stato è rappresentato da un'**autonomia**, un livello di **Intelligenza Artificiale** e di **tecnologia**. Un marziano può attaccare i terrestri.

Più in dettaglio…

- La **matricola** è una stringa di caratteri.
- L'**autonomia** è rappresentata da un indice compreso tra 1 e 100. Quando tale indice è 100 l'autonomia è massima, quando è zero il marziano è morto.
- L'**AI** è rappresentata da un indice compreso tra 1 e 10.
- La **tecnologia** è rappresentata da un indice compreso tra 1 e 10.

? *Come astrarre tali entità marziane?*

L'entità MARZIANO (2)

*Occorrerà una classe **Marziano** che definisce il tipo di dato astratto 'Marziano'.*

- **Parte Nascosta (stato):** matricola,AI,tecnologia e autonomia **non** devono essere accessibili direttamente dall'esterno…
- **Parte Visibile (costruttori):**
  - **Marziano**() //parametri di default: matricola = "indefinito"
    //automomia = 50; AI = 5; tecnologia=5
  - **Marziano** (String matricola, int[] param)
    // param è un array di 3 interi che corrispondono rispettivamente ai
    // valori di *autonomia*, *AI* e *tecnologia* di inizializzazione

Al termine della inizializzazione ogni costruttore deve stampare in uscita: "Il Marziano [matricola] entra in gioco."

L'entità MARZIANO (3)

- **Parte Visibile (metodi):**
  - void **attaccaTerrestri**() //vedi slide 11
  - boolean **attivo**() // ***true*** se autonomia>0; ***false*** altrimenti
  - String **toString**() // restituisce la matricola e il livello di autonomia
    // corrente es: "Marziano x102. Autonomia = 50"
  - boolean **equals** (Marziano x)

Ai fini del gioco due oggetti 'Marziani' sono uguali se hanno le stesse 'potenzialità offensive'. Ossia se, indipendentemente dalla matricola, hanno lo stesso livello di autonomia, AI e tecnologia. Se uno solo di questi parametri è diverso i due Marziani non si equivalgono.

L’entità TERRESTRE (1)

Ogni terrestre ha un **nome**. Il suo stato è rappresentato dalla **vita** che ha ancora a disposizione, dalla **velocità** di combattimento e da un **insieme di armi**. Un terrestre può attaccare i marziani.

Ogni arma è caratterizzata da un **nome**, da una **potenza**, e dal numero di **colpi** ancora disponibili. Quando un terrestre attacca i marziani può scegliere una delle armi disponibili. Ogni volta che l’arma viene usata, il numero di colpi disponibili è decrementato. Si noti che la possibilità di utilizzare l’arma è condizionale alla disponibilità di colpi.

L'entità TERRESTRE (2)

Più in dettaglio…

- Il **nome** è una stringa di caratteri.
- La **vita** disponibile è rappresentata da un indice compreso tra 1 e 100. 100 è l'indice massimo di vita disponibile, quando la vita è zero il terrestre è morto.
- La **velocita** è rappresentata da un indice compreso tra 1 e 10.
- L'**insieme di armi** è rappresentato da un array di oggetti che modellano l'**arma**.

? *Come astrarre le entità terrestri?*

L'entità TERRESTRE (3)

*Occorrerà una classe **Terrestre** che definisce il tipo di dato astratto 'Terrestre'.*

- **Parte Nascosta (stato):**

  Nome,vita,velocità e l'insieme di armi disponibili non devono essere accessibili direttamente dall'esterno…

- **Parte Visibile (costruttori):**

  — **Terrestre**() // parametri di default: nome = "sconosciuto"
  // vita = 50; velocita = 5; numero massimo armi =5

  — **Terrestre** (String nome, int vita, int velocita, int mxArmi)
  // param è un array di 3 interi che corrispondono ai valori di
  // inizializzazione di *vita*, *velocita* e numero massimo armi

Al termine della inizializzazione ogni costruttore deve stampare in uscita: "Il Terrestre [nome] entra in gioco."

L'entità TERRESTRE (3)

- **<u>Parte Visibile (operazioni)</u>:**
  - void **addArma**(Arma weapon)
  - void **attaccaMarziani**(int scelta)
  - boolean **vivente**() // ***true*** se vita>0; ***false*** altrimenti
  - String **toString**() // restituisce il nome e il livello di vita
    // corrente es: "Terrestre Ken. Vita = 75"
  - boolean **equals** (Terrestre x)

Ai fini del gioco due oggetti 'Terrestri' sono uguali se hanno le stesse 'potenzialità offensive'. Ossia se, <u>indipendentemente dal nome</u>, hanno lo stesso livello di vita, velocità e armamento. Se uno solo di questi parametri è diverso i due Terrestri non si equivalgono.

L’entità ARMA (1)

*Occorrerà una classe **Arma** che definisce il tipo di dato astratto ‘Arma’.*

- **<u>Parte Nascosta (stato)</u>:**
  Nome, potenza, e numero di colpi ancora disponibili non devono essere accessibili direttamente dall’esterno…
- **<u>Parte Visibile (costruttori)</u>:**
  - **Arma** (String name, int power, int colpi)

```
// name rappresenta il nome dell’arma, power rappresenta la
// potenza e colpi il numero di colpi ancora disponibili
```

L’entità ARMA (2)

- **Parte Visibile (operazioni):**
  - int **fire**() // restituisce un valore che rappresenta l’efficacia dell’arma pari al prodotto fra il numero dei colpi disponibili e la potenza. Restituisce 0 se il numero dei colpi rimasti è nullo.
  - Int **getColpi**()
  - int **getPower**()
  - String **getName**()
  - String **toString**()
  - boolean **equals** (Arma x) // restituisce true se sono uguali la potenza ed il numero di colpi ancora disponibili

L'applicazione 'GiocoBattaglia'(1)

Il componente software che gestisce il gioco dovrà…

1. Segnalare in uscita che il gioco ha inizio.
2. Definire e creare un oggetto 'Marziano' con matricola e parametri di default (costruttore senza parametri).
3. Definire e creare un oggetto 'Terrestre' con nome e parametri di default (costruttore senza parametri).
4. Definire e creare il marziano "x102" con autonomia=50, AI=5, tecnologia=5.
5. Definire e creare il terrestre "Ken" con vita=75, velocita=10, armamento=6.
6. Aggiungere al terrestre sconosciuto un'arma alabarda spaziale con potenza 0 e colpi 2.

L’applicazione ‘GiocoBattaglia’(2)

7. Aggiungere a Ken un’arma lancia rotante con potenza 15 e colpi 2.
8. Confrontare l’equivalenza ai fini del gioco dei due Terrestri creati (“Ken” e “sconosciuto”) e stampare in uscita un messaggio con il risultato del confronto.
9. Ripetere il punto 6 per i due Marziani creati.
10. Mostrare in uscita lo stato di vita di Ken e di autonomia di x102 con l’indicazione che si tratta dello stato prima dell’attacco.
11. Far attaccare (se vivi) 2 volte Ken e x102 e mostrare ogni volta in uscita il loro stato di vita e autonomia con l’indicazione che si tratta dello stato dopo dell’attacco.
12. Segnalare in uscita che il gioco è terminato.

Simulazione di un attacco

*Viene qui riportato il codice del metodo **attaccaMarziani(int scelta)** nella classe **Terrestre**. Si basa per semplicità sui valori delle variabili di istanza.*

Da utilizzare nella implementazione della classe **Terrestre** e da adattare per la classe **Marziano**.

```
/**
 * Simulazione di un attacco.
 */
public void attaccaMarziani(int scelta){ // attacca i marziani con l'arma
        if ((scelta<0) || (scelta>=numArmiDisponibili)) return;
        if (!vivente()) System.out.println("Non posso attaccare:Sono morto!");
        else {
              //algoritmo di attacco semplificato
              vita = vita - 20 + 2 * velocita + vet[scelta].fire();
              if (vita < 0) vita=0;
            }
}
```

Esercitazione n°2

Come procedere?

1. Scaricare il file **GiocoBattaglia.java** dal sito Web del corso e salvarlo in **C:\TEMP**

2. Il file **GiocoBattaglia.java** contiene il ***main()*** completo dell'applicazione.

Capirne il funzionamento e implementare le classi mancanti: **Terrestre.java** e **Marziano.java** sulla base della specifica data. Si usi come editor JCreator.

3. Compilare tutto ed eseguire l'applicazione usando i comandi della JDK attraverso la Console per Java.

Esercitazione n°2: facoltativo

- Documentare **(javadoc)** tutti i metodi e le variabili delle classi Marziano.java e Terrestre.java.

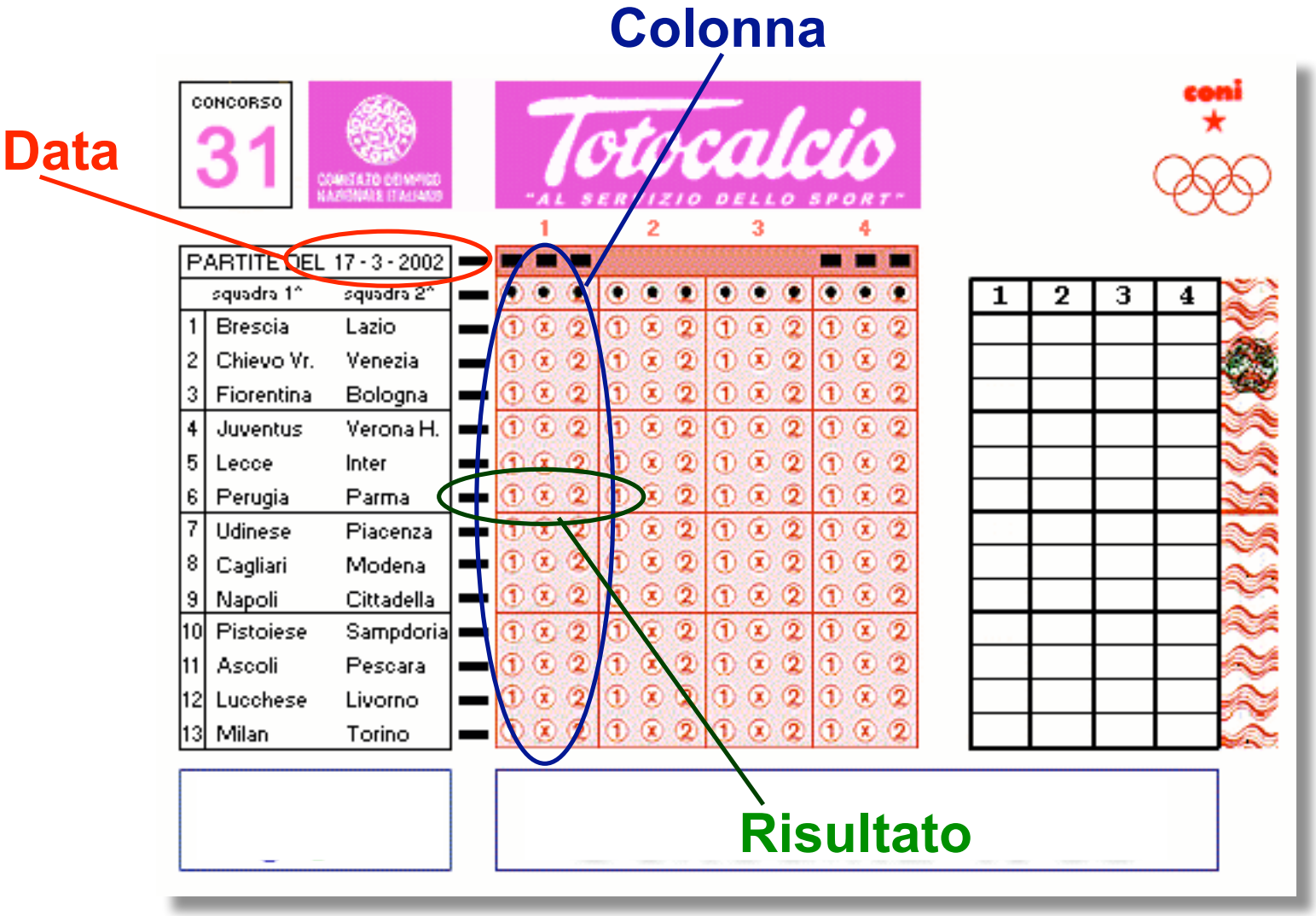
- Cambiare il componente software che gestisce il gioco:

  — Creare più personaggi.

- Varianti nei tipi di dati astratti:

  — Cambiare la visualizzazione dello stato (metodo toString()) degli oggetti stampando in uscita, per esempio, anche il valore corrente di AI e tecnologia per i Marziani e di velocità e armamento per i Terrestri.

Totocalcio


Colonna
Data
Risultato
CONCORSO
31
Totocalcio
"AL SERVIZIO DELLO SPORT"
coni
1 2 3 4
PARTITE DEL 17 - 3 - 2002
squadra 1^ squadra 2^
1 Brescia Lazio
2 Chievo Vr. Venezia
3 Fiorentina Bologna
4 Juventus Verona H.
5 Lecce Inter
6 Perugia Parma
7 Udinese Piacenza
8 Cagliari Modena
9 Napoli Cittadella
10 Pistoiese Sampdoria
11 Ascoli Pescara
12 Lucchese Livorno
13 Milan Torino
1 2 3 4

Totocalcio

Si richiede di progettare ed implementare l'ADT **schedina**.

Ogni **schedina** è caratterizzata da una **data** espressa nel formato *gg/mm/aaaa*. Inoltre ogni schedina è caratterizzata da **4 colonne** costituite da *13* **risultati**.

Per semplicità si assume che i risultati in una certa colonna possano essere esclusivamente **1**, **X** oppure **2**.

Si richiede di implementare i metodi

```
public String getColonna(int numColonna)
```

Che preso in ingresso l'identificatore della colonna desiderata restituisce in uscita la stringa che la rappresenta

Es. **`1X111XXX1X21X`**

Totocalcio

```
public boolean modifyColonna(int numCol,
                             Colonna col)
```

Che sostituisce la colonna identificata da **`numCol`** con la colonna specificata da **`col.`**

```
public int vittorieCasa(int numCol)
```

Calcola il numero di vittorie in casa (ovvero di “1”) pronosticate nella colonna identificata da **`numCol`**.

Totocalcio

Si progetti ed implementi un componente software GestoreSchedina che consente la gestione delle schedine di un utente. Il componente è caratterizzato dal nome dell'utente e da un array di oggetti Schedina.

```
public void insSchedina(Schedina sched)
```

Consente di inserire una nuova schedina. Anzitutto si controlla se è disponibile spazio nell'array che memorizza le schedine ed in caso affermativo la schedina identificata da sched viene inserita.

Totocalcio

`public void delSchedina(int giorno, int mese, int anno)`

Ricerca la schedina identificata dalla data specificata e se trovata la elimina dall'array che memorizza le schedine.

`public String toString()`

Restituisce in uscita una stringa secondo il formato

`[nome utente], [num Schedine] schedine`

Dove [nome utente] rappresenta il nome dell'utente mentre [num Schedine] il numero delle schedine che possiede

Totocalcio

Si crei un componente Main che implementa il main della applicazione

Si richiede di creare una schedina con una colonna

xx1x21x21122x

Si crei un gestore schedine per l'utente bart

Inserire la schedina nel gestore.