Programmazione Orientata agli Oggetti in Linguaggio Java

Ruoli e Responsabilità: Conclusioni

versione 1.3

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Riepilogo
- Diagrammi UML
- API di Java
  - ⇨Le Collezioni
  - ⇨Tipi Generici
  - ⇨java.util.Random
  - ⇨String e StringBuffer
  - ⇨Enumerazioni

Riassumendo

- Interfaccia
  - ⇨tutto quello che un componente sa fare
  - ⇨definisce i compiti del componente
- Implementazione
  - ⇨tutti i dettagli di come lo fa (gli strumenti e le tecniche per svolgere i suoi compiti)
- Analogo
  - ⇨massaia ed idraulico

Riassumendo

- Obiettivo
  - ⇨costruire società di componenti organizzati
  - ⇨i componenti si dividono i compiti e collaborano secondo regole precise
- Ruolo e Strato applicativo
  - ⇨il ruolo rappresenta il mestiere del componente
  - ⇨lo strato applicativo rappresenta il luogo di lavoro del componente

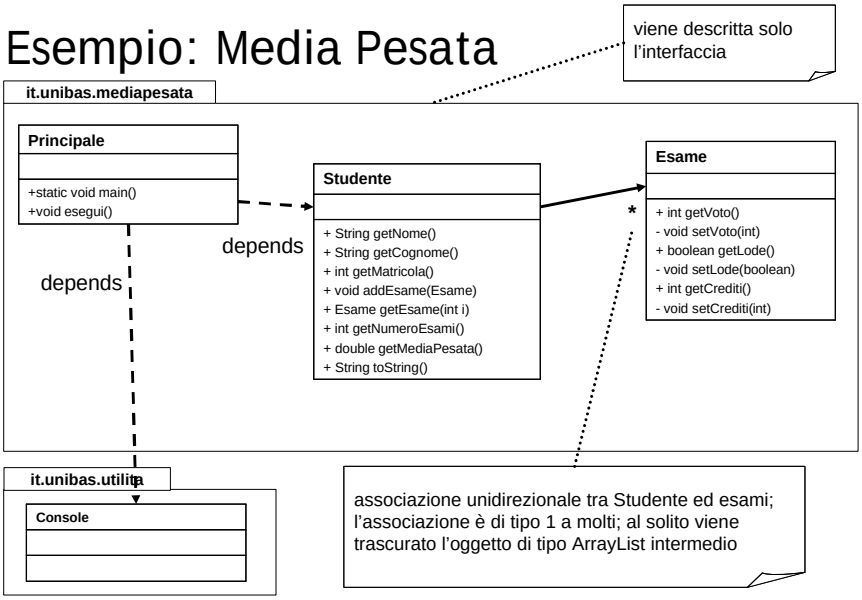
Riassumendo

- Analogo
  - ⇨la lavanderia
- Interfaccia e controllo
  - ⇨compito di relazione con il pubblico
- Modello e persistenza
  - ⇨compito di fare funzionare la bottega
- Servizi di base
  - ⇨infrastruttura e strumenti di lavoro

Diagrammi UML

- In questo argomento
  - ⇨la media pesata
  - ⇨indovina il numero
  - ⇨la morra cinese
- Diagrammi di documentazione
  - ⇨riconsideriamo i diagrammi UML degli esempi
  - ⇨cominciamo dai diagrammi delle classi e poi vediamo alcuni diagrammi di collaborazione

Esempio: Media Pesata



Diagrammi UML

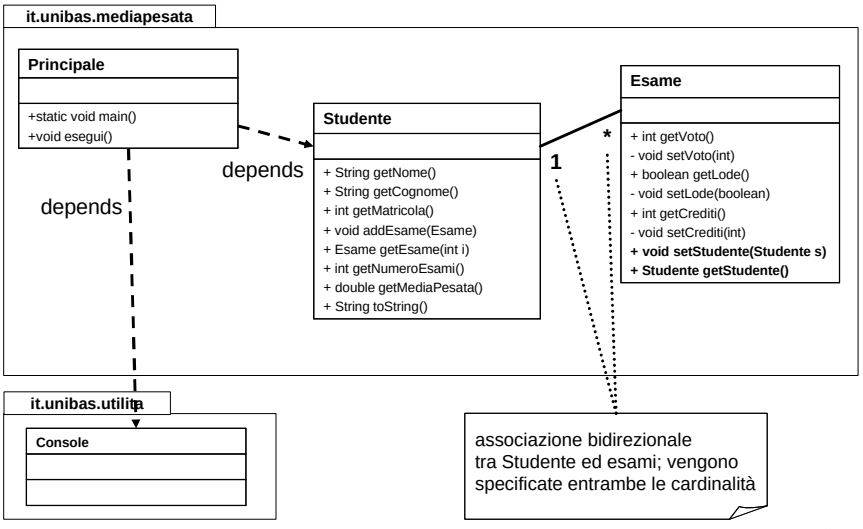
- Nota
  - ⇨l'associazione tra Studente ed Esame è unidirezionale (lo studente mantiene la lista di riferimenti agli esami, ma non il viceversa)
- Ma, anche in questo caso
  - ⇨l'associazione avrebbe potuto essere bidirezionale
  - ⇨es: in Studente una lista di Esami, in Esame un riferimento allo studente

Diagrammi UML

```
public class Studente {
   private String nome, cognome;
   private int matricola;
   private java.util.ArrayList esami = new java.util.ArrayList();
   ...
}

public class Esame {
   private String insegnamento;
   private int vodo, crediti;
   private boolean lode;
   private Studente studente;
   ...
}
```

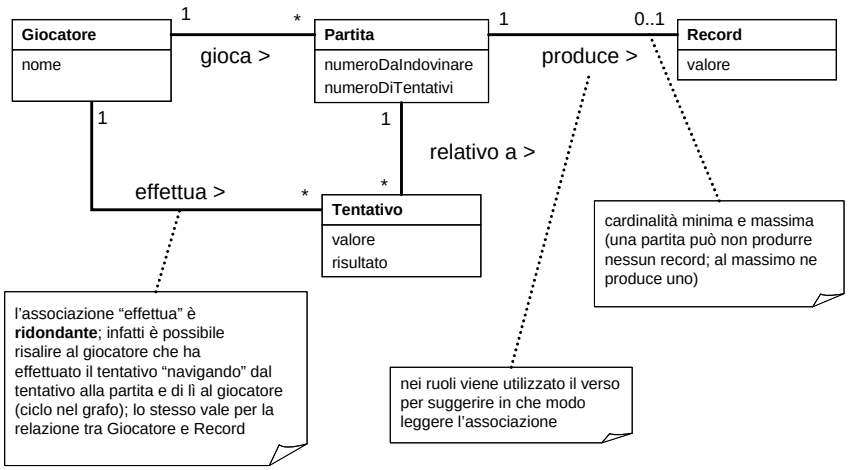

Media Pesata: Diagramma delle Classi



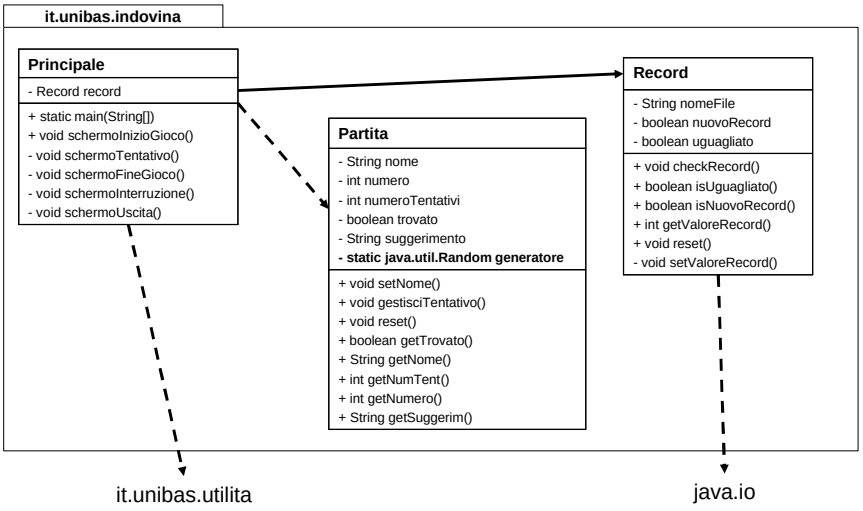
Diagrammi UML

- Indovina il numero
  - ⇨di seguito vediamo prima un possibile diagramma concettuale
  - ⇨poi il diagramma delle classi finale dell'appl.
  - ⇨e alcuni diagrammi di collaborazione
- Al solito
  - ⇨le classi del modello dell'applicazione discendono dal diagramma concettuale
  - ⇨non tutti i concetti sono diventati classi

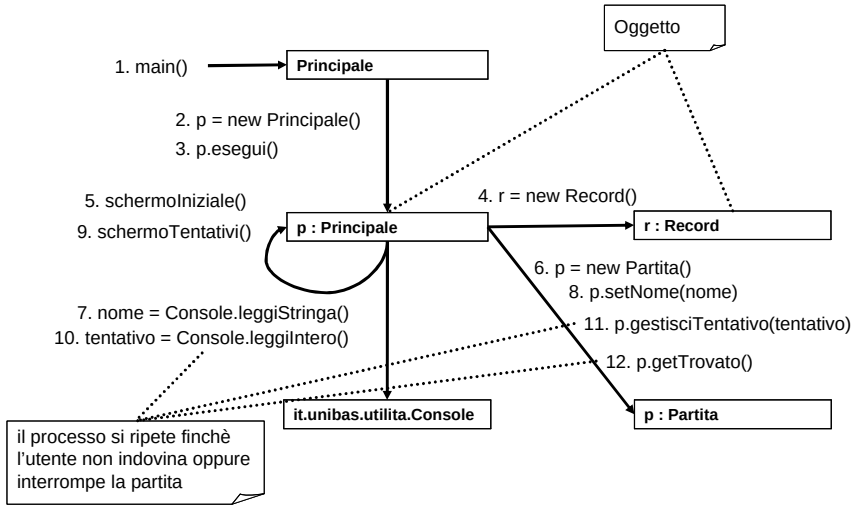
Indovina il Numero

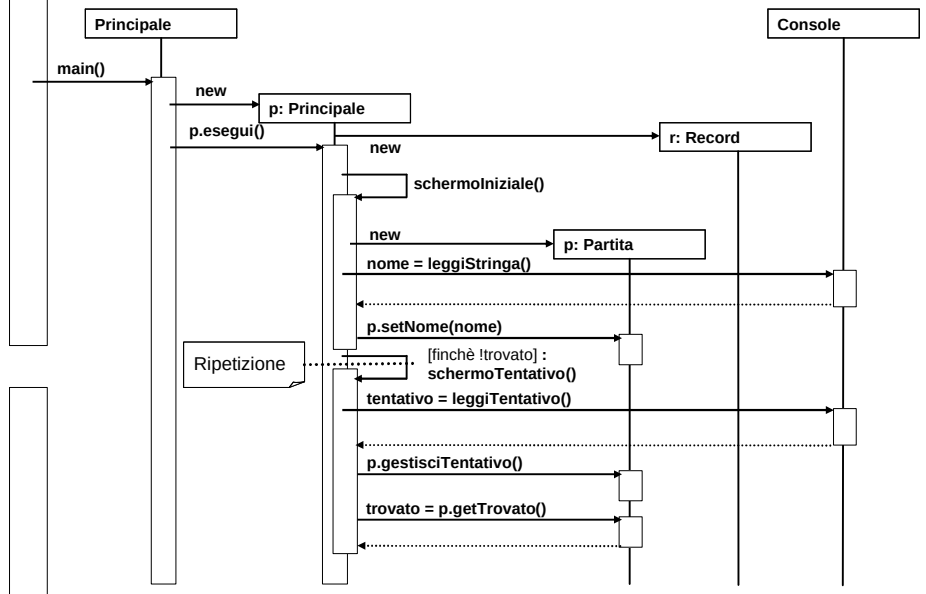


Indovina il Numero



Indovina il Numero: Collaborazione

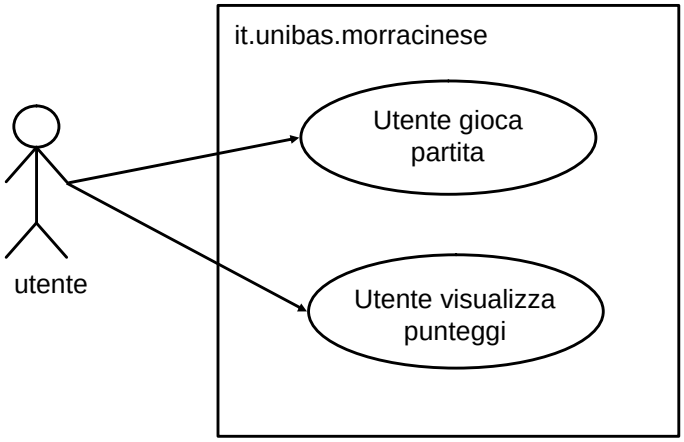




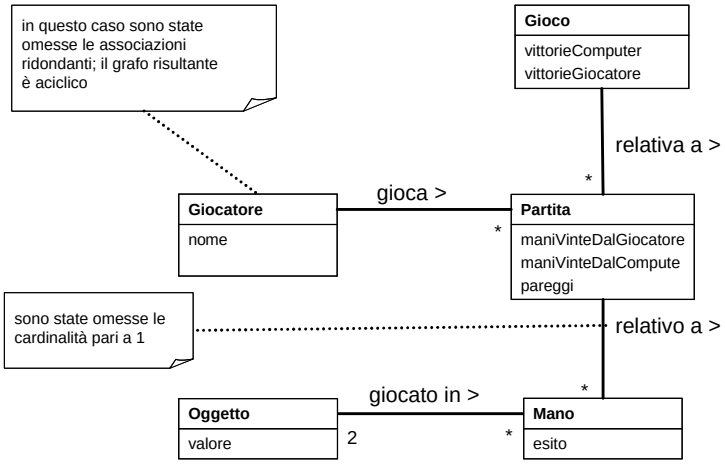
Diagrammi UML

- La Morra Cinese
  - ⇨il diagramma dei casi d'uso
  - ⇨un possibile diagramma concettuale
  - ⇨poi il diagramma delle classi finale dell'applicazione
  - ⇨un diagramma di collaborazione

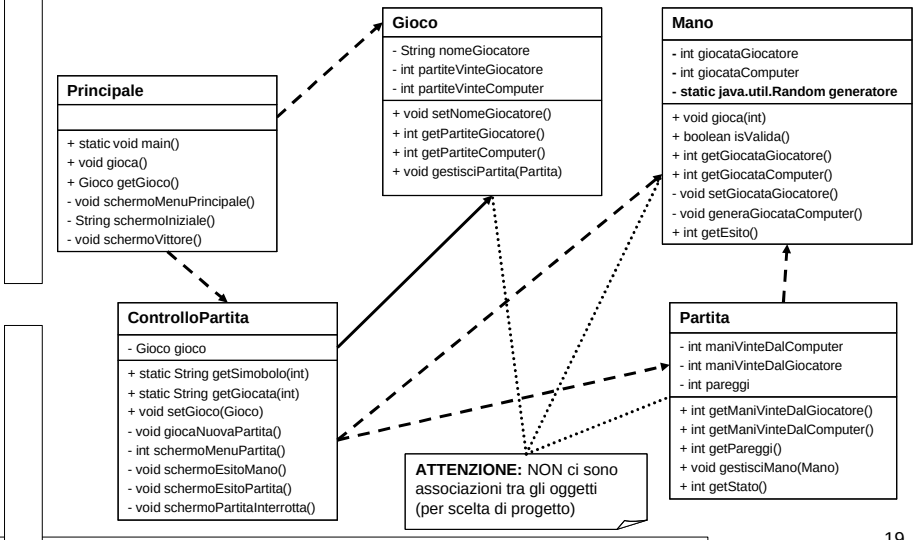
La Morra Cinese



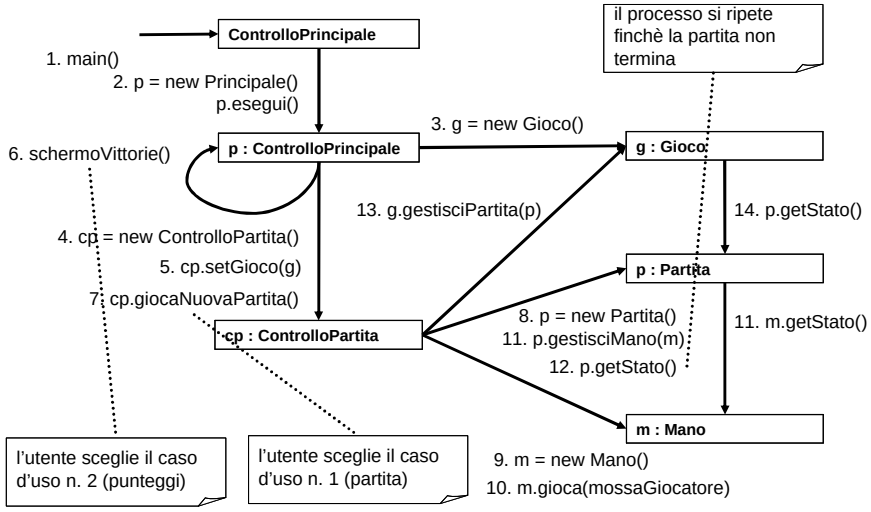
La Morra Cinese



La Morra Cinese



La Morra Cinese: Collaborazione



La Piattaforma Java

>> java.util in Jdk1.4

- Package e classi notevoli
  - ⇨il package java.util
  - ⇨le collezioni
  - ⇨java.util.Random
  - ⇨java.lang.StringBuffer
- Il package java.util
  - ⇨è noto principalmente come il package delle collezioni di Java

Le Collezioni

- Le collezioni principali di java.util
  - ⇨java.util.List (interface)
  - ⇨java.util.ArrayList
  - ⇨java.util.LinkedList
- Le vecchie collezioni
  - ⇨java.util.Stack
  - ⇨java.util.Vector: precedente implementazione di ArrayList, viene considerata “deprecata”

Le Collezioni

- Altre collezioni (>>)
  - ⇨java.util.TreeSet: insieme (collezione non ordinata priva di duplicati) rappresentato sotto forma di albero
  - ⇨java.util.HashMap: mappa associativa
  - ⇨java.util.SortedMap: mappa ordinata rappresentata sotto forma di albero

Le Collezioni

- La filosofia delle collezioni di J2SE 1.4
  - ⇨è necessario fornire collezioni che consentano un utilizzo con tipi diversi
  - ⇨la soluzione di J2SE fino a 1.4 è fissare il tipo della collezione (collezione di riferimenti a Object)
  - ⇨ma scegliendo un tipo degli elementi sufficientemente “generico” da consentire di inserire riferimenti ad oggetti qualsiasi

Le Collezioni

- java.util.ArrayList
  - ⇨lista rappresentata con array e indicatore di riempimento
  - ⇨limite tradizionale: dimensione fissata dell'array
  - ⇨in questo caso questo limite non esiste
- Com'è possibile ?
  - ⇨"trucco" collegato all'utilizzo degli oggetti

Le Collezioni

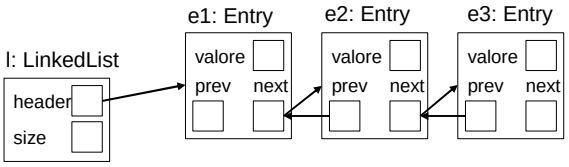
>> ArrayList.java

- In sintesi
  - ⇨alla creazione, la classe crea un array di 10 riferimenti ad Object
  - ⇨quando l'array si riempie, crea un nuovo array di dimensione 1.5 volte la precedente
  - ⇨copia tutti i vecchi riferimenti nel nuovo array
  - ⇨sostituisce il vecchio array con il nuovo (cambiando il valore del riferimento)

Le Collezioni

- java.util.LinkedList
  - ⇨lista con rappresentazione doppiamente collegata
  - ⇨ma in Java non ci sono i puntatori
- Com'è possibile ?
  - ⇨al posto dei puntatori vengono utilizzati i riferimenti
  - ⇨puntatori e riferimenti sono intercambiabili

Le Collezioni



- In sintesi
  - ⇨viene utilizzata una ulteriore classe Entry
  - ⇨ciascun elemento della lista è un oggetto di tipo Entry con valore, riferimento all'elemento successivo, riferimento all'elemento precedente
  - ⇨la lista contiene un riferimento all'oggetto di tipo Entry che rappresenta la testa
  - ⇨nota: manca il puntatore alla coda (>>)

Le Collezioni

>> java.util.Arrays

- La classe java.util.Arrays
  - ⇨vari metodi statici che implementano operazioni notevoli sugli array
  - ⇨es: equals (confronto tra array)
  - ⇨es: binarySearch (ricerca binaria)
  - ⇨es: sort (ordinamento)
- Un altro metodo utile
  - ⇨System.arraycopy: per copiare un array in un altro

ATTENZIONE
tipi generici
J2SE 5.0

Tipi Generici

- Una nota importante
  - ⇨il rilascio di J2SE 1.5 ha completamente cambiato l'approccio alle collezioni
  - ⇨sono stati infatti introdotti i tipi generici
- Tipo generico
  - ⇨tipo parzialmente specificato
  - ⇨il nome del tipo contiene un "parametro"
  - ⇨per completare la sua definizione è necessario specificare un argomento

Tipi Generici

- Nota
  - ⇨il parametro rappresenta a sua volta un tipo (cosa del tutto non convenzionale)
  - ⇨e quindi anche l'argomento
- Sintatticamente
  - ⇨i tipi parametro sono inclusi tra < >
- Esempio di dichiarazione di tipo generico

ArrayList<T> ← T è un parametro che rappresenta un tipo

Tipi Generici

- Idea dei generici
  - ⇨nella dichiarazione viene specificato il parametro (tipo generico)
  - ⇨all'utilizzo il programmatore fornisce l'argomento (tipo specifico)

tipo parametro
tipo argomento

```
public class ArrayList<T> {
    private T arrayElementi[];
    public ArrayList<T>() {...}
    public T get(int i) {...}
}
```

```
// in Studente
ArrayList<Esame> esami =
        new ArrayList<Esame>();
```

Tipi Generici

- Fornendo l'argomento
  - ⇨il codice della classe è come se fosse riscritto sostituendo T con Esame

```
public class ArrayList<T> {
    private T arrayElementi[];
    public ArrayList<T>() {...}
    public T get(int i) {...}
}
```

→

```
public class ArrayList<Esame> {
    private Esame arrayElementi[];
    public ArrayList<Esame>() {...}
    public Esame get(int i) {...}
}
```

```
// in Studente
ArrayList<Esame> esami = new ArrayList<Esame>();
```


Tipi Generici

- La filosofia di J2SE 1.5
  - ⇨l'obiettivo è sempre fornire collezioni che consentano un utilizzo con tipi diversi
  - ⇨la soluzione di J2SE fino a 1.5 è utilizzare tipi generici per le collezioni (ovvero tipi parzialmente specificati)
  - ⇨chiedendo poi al programmatore di definire il tipo specifico al momento dell'utilizzo della collezione

Tipi Generici

- Vantaggio n. 1
  - ⇨primo vantaggio: avendo le collezioni un tipo definito è possibile omettere i cast quando si estraggono gli oggetti

```
// senza generici
ArrayList esami =
        new ArrayList();
esami.add(new Esame());
Esame e = (Esame)esami.get(0);
```

```
// con i generici
ArrayList<Esame> esami =
        new ArrayList<Esame>();
esami.add(new Esame());
Esame e = esami.get(0);
```


Tipi Generici

- Vantaggio n. 2
  - ⇨il compilatore è in grado di effettuare tutti i controlli sui tipi a tempo di compilazione
  - ⇨si evitano errori a tempo di esecuzione

```
// senza generici
ArrayList esami = new ArrayList();
esami.add(new Esame());
Studente s = (Studente)esami.get(0);
```

viene sollevato un errore di tipo ClassCastException a tempo di esecuzione

```
// con i generici
ArrayList<Esame> esami =
        new ArrayList<Esame>();
esami.add(new Esame());
Studente s = esami.get(0);
```

il compilatore segnala l'errore di tipo a tempo di compilazione

Tipi Generici

- Nota
  - ⇨in effetti, è un comportamento molto simile a quello degli array
  - ⇨in effetti, il tipo array di Java è sempre stato un tipo generico
  - ⇨il cui tipo delle componenti viene specificato alla creazione dell'array
  - ⇨es: String[] vettore = new String[];

Tipi Generici

>> java.util in Jdk1.5

- Il package java.util in J2SE 1.5
  - ⇨è disseminato di tipi generici
- In altri termini
  - ⇨i generici sono considerati lo strumento standard della nuova versione di Java
  - ⇨anche se, per ragioni di compatibilità, è ancora possibile utilizzare le vecchie collezioni di J2SE 1.4

Tipi Generici

- Non utilizzando i generici
  - ⇨il compilatore segnala dei messaggi di avvertimento durante la compilazione
- Unchecked Warning
  - ⇨avvertimento che c'è un'operazione per cui il controllo di tipo non è perfetto
  - ⇨es: utilizzo di una collezione tradizionale

```
Note: ProvaGenerici.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```


Tipi Generici

ATTENZIONE
all'approccio ai tipi generici

- In realtà, però...
  - ⇨al di là di questa presentazione superficiale, i generici hanno una semantica complessa
  - ⇨non sono utilizzabili sulle vecchie macchine virtuali
  - ⇨i vantaggi non sono straordinari nella maggior parte delle applicazioni
  - ⇨tutto sommato quindi possono essere evitati
  - ⇨in questo corso non verranno utilizzati oltre

La Classe java.util.Random

- La classe java.util.Random
  - ⇨costruisce generatori di numeri casuali
- In particolare
  - ⇨il costruttore crea un oggetto che è un generatore di numeri casuali inizializzandone il seme con il valore dell'orologio
  - ⇨per ottenere la sequenza pseudo-casuale: int nextInt(int max)

La Classe java.util.Random

- Attenzione all'utilizzo
  - ⇨due oggetti di tipo Random creati l'uno dopo l'altro avranno lo stesso valore del seme
  - ⇨e genereranno la stessa sequenza
- Esempio

```
public static void main(String[] args) {
    java.util.Random gen1 = new java.util.Random();
    java.util.Random gen2 = new java.util.Random();
    System.out.println(gen1.nextInt() + " " + gen2.nextInt());
}
```

i numeri generati saranno identici

La Classe java.util.Random

- Di conseguenza
  - ⇨è opportuno creare un unico generatore una volta per tutte
  - ⇨e utilizzarlo tutte le volte che serve
  - ⇨una buona pratica: generatori statici
- Un metodo alternativo
  - ⇨System.Math.random()

La Classe java.util.Random

```
package it.unibas.indovinasemplice;

public class Partita {

    private static java.util.Random generatore = new java.util.Random();
    private String nome; private String suggerimento;
    private int numeroDaIndovinare; private int numeroDiTentativi;
    private boolean trovato;

    public Partita() {
        this.numeroDaIndovinare = Math.abs(Partita.generatore.nextInt(100) + 1);
        this.trovato = false;
        this.numeroDiTentativi = 0;
        this.suggerimento = "Ho scelto un numero tra 1 e 100. Prova a indovinarlo.";
    }
```

La Classe java.util.Random

```
package it.unibas.morracinesesemplice;

public class Mano {

   private static java.util.Random generatore = new java.util.Random();

    private void generaGiocataComputer() {
        this.giocataComputer = Math.abs(Mano.generatore.nextInt(3) + 1);
    }

   ...

}
```


String e StringBuffer

- java.lang.String
  - ⇨rappresenta stringhe immutabili
  - ⇨ogni volta che concateno stringhe (+) di fatti sto costruendo nuove stringhe
- In alcuni casi
  - ⇨è necessario costruire stringhe concatenando assieme molti pezzi
  - ⇨in questi casi può essere opportuno utilizzare uno StringBuffer invece che String

String e StringBuffer

>> java.lang.StringBuffer

- StringBuffer
  - ⇨classe i cui oggetti rappresentano stringhe modificabili
- I principali metodi
  - ⇨append(): concatena alla stringa un'altra stringa
  - ⇨setCharAt(int i): cambia il carattere i-esimo
  - ⇨insert(): inserisce una stringa in un'altra

```
private static String getSimbolo(int mossa) {
      StringBuffer simbolo = new StringBuffer();
      if (mossa == Mano.CARTA) {
         simbolo.append(" _____  \n");
         simbolo.append("| ___ \\ \n");
         simbolo.append("| ___ | \n");
         simbolo.append("| ___ | \n");
         simbolo.append("|_____| \n");
      } else if (mossa == Mano.FORBICI) {
         simbolo.append(" \\\\    // \n");
         simbolo.append("  \\\\  //  \n");
         simbolo.append("   \\\\//   \n");
         simbolo.append("   //\\\\   \n");
         simbolo.append("  //  \\\\  \n");
         simbolo.append(" ()    ()   \n");
      } else if (mossa == Mano.SASSO) {
         simbolo.append("  ___   \n");
         simbolo.append(" (   )  \n");
         simbolo.append("( (   ) \n");
         simbolo.append("(____)  \n");
      }
      return simbolo.toString();
   }
```

Enumerazioni

- Un'ultima annotazione
  - ⇨rappresentazione dei valori di tipi enumerabili nella Morra Cinese (oggetto ed esito)
- La soluzione tradizionale di Java
  - ⇨costanti intere
- Esempio: nella classe Gioco

```
public final static int CARTA = 1;
public final static int FORBICI = 2;
public final static int SASSO = 3;
```


Enumerazioni

- Vantaggi di questo approccio
  - ⇨molto semplice da utilizzare (es: acquisire i dati, scrivere il codice)
- Svantaggi di questo approccio
  - ⇨il compilatore non effettua nessuna verifica sui valori della giocata
  - ⇨qualsiasi valore intero è in linea di principio accettabile come valore della giocata

Enumerazioni

ATTENZIONE
alle enumerazioni
J2SE 5.0

- A partire da J2SE 1.5
  - ⇨è possibile definire in Java enumerazioni
- Enumerazione (“Typesafe Enum”)
  - ⇨classe Java semplificata
  - ⇨i cui membri sono esclusivamente costanti
  - ⇨ciascuna costante è un oggetto
  - ⇨trasformabile in stringa con toString()
  - ⇨e in intero con ordinal()

Enumerazioni

- Esempio
  - ⇨una enumerazione per l’oggetto giocato
  - ⇨public enum Oggetto {CARTA, FORBICI, SASSO};
- Semantica
  - ⇨definisce un nuovo tipo (Oggetto)
  - ⇨i cui membri costanti sono i riferimenti di tipo Oggetto Oggetto.CARTA, Oggetto.FORBICI, Oggetto.SASSO
  - ⇨Oggetto.CARTA.toString() -> “CARTA”
  - ⇨Oggetto.CARTA.ordinal() -> 0

Enumerazioni

essendo una classe, se è contenuta nel file Mano.java non può essere pubblica ma deve essere friendly; alternativa: in Oggetto.java

```
package it.unibas.morracinese;

enum Oggetto {CARTA, FORBICI, SASSO};

public class Mano {

    private Oggetto giocataGiocatore;
    private Oggetto giocataComputer;

    public boolean isValida(int giocata) {
        return (giocata >= Mano.CARTA.ordinal()
                && giocata <= Mano.SASSO.ordinal());
    }

    ...
```


Enumerazioni

- Attenzione
  - ⇨enum è una parola chiave nuova introdotta in J2SE 5.0 (prima non lo era)
  - ⇨riduce la compatibilità con il codice esistente (es: classe che ha una proprietà o variabile chiamata “enum”)
- Per risolvere questo problema
  - ⇨è necessario compilare il codice con un’opzione particolare

Enumerazioni

- Il livello dei sorgenti
  - ⇨opzione –source es: javac –source 1.5 Mano.java
  - ⇨valore standard: 1.3
- Se il livello è 1.3
  - ⇨enum non è una parola riservata, ma non è consentito usare enumerazioni
- Se il livello è 1.5
  - ⇨enum è una parola riservata, non è consentito usarla come identificatore

Riassumendo

- Riepilogo
- Diagrammi UML
- API di Java
  - ⇨Le Collezioni
  - ⇨Tipi Generici
  - ⇨java.util.Random
  - ⇨String e StringBuffer
  - ⇨Enumerazioni

Termini della Licenza