Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 22 settembre 2014

TEMPO DISPONIBILE: 2 ore

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Number`: ogni oggetto della classe rappresenta un valore numerico. La classe possiede un costruttore privo di argomenti. Nelle librerie standard alcune classi involucro (ad esempio `Integer`, `Long`, `Float` e `Double`) sono definite estendendo `Number`.
- Classe `ElencoNumeri`: ogni oggetto della classe rappresenta un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int quantiPositivi()`
    Restituisce il numero di oggetti presenti nell'elenco che rappresentano valori positivi.
  - `public int quanti()`
    Restituisce il numero totale di oggetti presenti nell'elenco.
  - `public double frequenzaPositivi()`
    Restituisce un valore di tipo `double` uguale alla *frequenza* dei valori positivi nell'elenco, definita come il rapporto tra il numero di valori positivi e il numero totale di valori presenti nell'elenco. Ad esempio, se l'elenco contiene `10` valori e `3` di questi sono di tipo positivi, la frequenza è `0.3` (cioè `3` diviso `10`).
  - `public Double sommaDoublePositivi()`
    Restituisce il riferimento a un oggetto di tipo `Double` che rappresenta la somma di tutti i valori positivi di tipo `Double` presenti nell'elenco.

1. Scrivete l'implementazione del metodo `frequenzaPositivi`, *senza conoscere* l'implementazione di `ElencoNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'elenco contenga almeno un elemento.

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che in caso di elenco vuoto venga sollevata una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

3. La classe `ElencoNumeri` è implementata mediante un unico campo

```
private Number[] valori
```

che si riferisce ad un array contenente i valori presenti nell'elenco.
Scrivete l'implementazione del metodo `sommaDoublePositivi`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori aritmetici e di confronto a riferimenti di tipo* `Double` *(non* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`, che possiede *un unico costruttore*. Il costruttore riceve un argomento di tipo `int`. Tra i metodi di `Alfa` vi è `public int intValue()` che restituisce il valore specificato al momento della creazione dell'oggetto. Ad esempio, il metodo `intValue()` di un oggetto costruito invocando `new Alfa(10)` restituisce `10`. Nella classe `Number` il metodo `intValue()` è astratto.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int z;
  private static String x = "gatto";

  public Beta(int t) {
    super(t / 2);
    z = t * 2;
    x = x + x.length();
  }

  public int intValue() {
    return super.intValue() + z;
  }

  public static int getStatico() {
    return x.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(9);
    System.out.println(a.intValue());       //2
    a = new Alfa(4);
    System.out.println(a.intValue());       //3
    System.out.println(Beta.getStatico());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi degli esercizi precedenti, considerate una classe concreta `Gamma` che estende `Alfa` e implementa un'interfaccia `In`, e una classe *astratta* `Delta` che estende `Number`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ `Number` è un supertipo di `In`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ `Delta` è priva di costruttore
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)
- ☐ `Gamma` può contenere metodi astratti
- ☐ `Number` è un supertipo di `Beta`
- ☐ Il costruttore di `Gamma` richiama quello di `In`
- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`

b. Considerate le seguenti dichiarazioni di variabile: `Number n, Alfa a, Beta b, Gamma g, Delta d, In i`
Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente

- ☐ `d = (In) g`
- ☐ `i = n`
- ☐ `n = i`
- ☐ `i = d`
- ☐ `g = (Beta) a`
- ☐ `n = g`
- ☐ `g = n`
- ☐ `i = (Gamma) i`
- ☐ `b = n`
- ☐ `g = (In) g`

c. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ I campi statici appartengono agli oggetti
- ☐ La dichiarazione `int[] x` serve per *costruire* un array di valori di tipo `int`
- ☐ La Java Virtual Machine produce il bytecode
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ In Java l'espressione `7 / 3` è di tipo `double`
- ☐ Per risparmiare tempo e fatica, quando si digita un programma conviene scrivere `double` al posto di `Double` in quanto non c'è alcuna differenza di significato
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ In Java una classe può implementare direttamente più interfacce
- ☐ Le interfacce possono contenere campi statici
- ☐ In Java, il tipo dell'espressione `x / y` dipende dai valori contenuti nelle variabili `x` e `y`
- ☐ L'overriding di un metodo viene risolto in base al tipo del riferimento utilizzato nella chiamata
- ☐ In una stessa classe è possibile definire più metodi con la stessa stessa segnatura, ma tipo restituito differente
- ☐ Una variabile di tipo `Object` può memorizzare un riferimento a un oggetto qualsiasi
- ☐ I record di attivazione contengono i dati degli oggetti
- ☐ In Java gli array contenenti valori di tipo `int` sono oggetti
- ☐ Il tipo `Integer` di Java è primitivo
- ☐ Le interfacce sono prive di metodi concreti
- ☐ Rimuovendo "`super.`" nel codice del metodo `intValue` della classe `Beta` si ha un errore in compilazione
- ☐ Rimuovendo "`super.`" nel codice del metodo `intValue` della classe `Beta` si ha un errore in esecuzione
- ☐ Il compilatore Java produce il bytecode

6. Considerate la dichiarazione di variabile `String[] nomi` e il seguente frammento di codice:

```
int x = 2;
try {
  do
    x = 1 + x / nomi[x].length();
  while (x / x == x);
} catch (ArithmeticException e) {
  x = x + 12;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 49;
} catch (NullPointerException e) {
  x = x + 22;
}
```

Ricordate che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `""` indica la stringa vuota.

Per ognuno dei seguenti casi:

- *se l'esecuzione termina* scrivete nel riquadro corrispondente il valore della variabile `x` dopo l'esecuzione,
- *se l'esecuzione non termina* scrivete nel riquadro il simbolo ∞ o la parola "infinito".

(a) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"ape"`, `""`.

(b) `nomi` contiene `null`.

(c) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `""`, `"cane"`.

(d) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"", "bue"`, `"cane"`, `"ape"`.

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito dalle chiamate indicate nei due riquadri:

```
... int f(int x) {
  if (x <= 1)
    return 2;
  else
    return x * f(x / 2) + 3;
}
```

f(3)	f(6)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 22 settembre 2014

TEMPO DISPONIBILE: 2 ore

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Number`: ogni oggetto della classe rappresenta un valore numerico. La classe possiede un costruttore privo di argomenti. Nelle librerie standard alcune classi involucro (ad esempio `Integer`, `Long`, `Float` e `Double`) sono definite estendendo `Number`.
- Classe `ElencoNumeri`: ogni oggetto della classe rappresenta un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int quantiPositivi()`
    Restituisce il numero di oggetti presenti nell'elenco che rappresentano valori positivi.
  - `public int quanti()`
    Restituisce il numero totale di oggetti presenti nell'elenco.
  - `public double frequenzaPositivi()`
    Restituisce un valore di tipo `double` uguale alla *frequenza* dei valori positivi nell'elenco, definita come il rapporto tra il numero di valori positivi e il numero totale di valori presenti nell'elenco. Ad esempio, se l'elenco contiene `10` valori e `3` di questi sono di tipo positivi, la frequenza è `0.3` (cioè `3` diviso `10`).
  - `public Double sommaDoublePositivi()`
    Restituisce il riferimento a un oggetto di tipo `Double` che rappresenta la somma di tutti i valori positivi di tipo `Double` presenti nell'elenco.

1. Scrivete l'implementazione del metodo `frequenzaPositivi`, *senza conoscere* l'implementazione di `ElencoNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'elenco contenga almeno un elemento.

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che in caso di elenco vuoto venga sollevata una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

3. La classe `ElencoNumeri` è implementata mediante un unico campo

```
private Number[] valori
```

che si riferisce ad un array contenente i valori presenti nell'elenco.
Scrivete l'implementazione del metodo `sommaDoublePositivi`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori aritmetici e di confronto a riferimenti di tipo* `Double` *(non* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`, che possiede *un unico costruttore*. Il costruttore riceve un argomento di tipo `int`. Tra i metodi di `Alfa` vi è `public int intValue()` che restituisce il valore specificato al momento della creazione dell'oggetto. Ad esempio, il metodo `intValue()` di un oggetto costruito invocando `new Alfa(10)` restituisce `10`. Nella classe `Number` il metodo `intValue()` è astratto.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int z;
  private static String x = "topo";

  public Beta(int t) {
    super(t / 2);
    z = t * 2;
    x = x + x.length();
  }

  public int intValue() {
    return super.intValue() + z;
  }

  public static int getStatico() {
    return x.length();
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(7);
    System.out.println(a.intValue());       //2
    a = new Alfa(3);
    System.out.println(a.intValue());       //3
    System.out.println(Beta.getStatico());  //4
  }
}
```

Scrivete in ogni riquadro l'output prodotto dall'istruzione di stampa seguita dal commento indicato:

//1	//2	//3	//4

5. Oltre alle classi degli esercizi precedenti, considerate una classe concreta `Gamma` che estende `Alfa` e implementa un'interfaccia `In`, e una classe *astratta* `Delta` che estende `Number`.

a. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ `Number` è un supertipo di `Beta`
- ☐ `Number` è un supertipo di `In`
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ Il costruttore di `Gamma` richiama quello di `In`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` può contenere metodi astratti
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)
- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Delta` è priva di costruttore

b. Considerate le seguenti dichiarazioni di variabile: `Number n, Alfa a, Beta b, Gamma g, Delta d, In i`
Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente

- ☐ `i = (Gamma) i`
- ☐ `b = n`
- ☐ `i = d`
- ☐ `i = n`
- ☐ `n = i`
- ☐ `d = (In) g`
- ☐ `n = g`
- ☐ `g = (In) g`
- ☐ `g = n`
- ☐ `g = (Beta) a`

c. Nel riquadro che precede ciascuna affermazione, scrivete V se l'affermazione è vera, F se è falsa:

- ☐ In Java una classe può implementare direttamente più interfacce
- ☐ L'overriding di un metodo viene risolto in base al tipo del riferimento utilizzato nella chiamata
- ☐ In Java gli array contenenti valori di tipo `int` sono oggetti
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ Le interfacce sono prive di metodi concreti
- ☐ Rimuovendo "`super.`" nel codice del metodo `intValue` della classe `Beta` si ha un errore in esecuzione
- ☐ Il compilatore Java produce il bytecode
- ☐ In Java l'espressione `7 / 3` è di tipo `double`
- ☐ La dichiarazione `int[] x` serve per *costruire* un array di valori di tipo `int`
- ☐ Il tipo `Integer` di Java è primitivo
- ☐ Le interfacce possono contenere campi statici
- ☐ In Java, il tipo dell'espressione `x / y` dipende dai valori contenuti nelle variabili `x` e `y`
- ☐ La Java Virtual Machine produce il bytecode
- ☐ In una stessa classe è possibile definire più metodi con la stessa stessa segnatura, ma tipo restituito differente
- ☐ Una variabile di tipo `Object` può memorizzare un riferimento a un oggetto qualsiasi
- ☐ I record di attivazione contengono i dati degli oggetti
- ☐ Per risparmiare tempo e fatica, quando si digita un programma conviene scrivere `double` al posto di `Double` in quanto non c'è alcuna differenza di significato
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ Rimuovendo "`super.`" nel codice del metodo `intValue` della classe `Beta` si ha un errore in compilazione
- ☐ I campi statici appartengono agli oggetti

6. Considerate la dichiarazione di variabile `String[] nomi` e il seguente frammento di codice:

```
int x = 2;
try {
  do
    x = 1 + x / nomi[x].length();
  while (x / x == x);
} catch (ArithmeticException e) {
  x = x + 15;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 37;
} catch (NullPointerException e) {
  x = x + 46;
}
```

Ricordate che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenti di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenti di accedere a un oggetto tramite un riferimento `null`,
- `""` indica la stringa vuota.

Per ognuno dei seguenti casi:

- *se l'esecuzione termina* scrivete nel riquadro corrispondente il valore della variabile `x` dopo l'esecuzione,
- *se l'esecuzione non termina* scrivete nel riquadro il simbolo ∞ o la parola "infinito".

(a) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `""`, `"bue"`, `"cane"`, `"ape"`. ☐

(b) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `""`, `"cane"`. ☐

(c) `nomi` contiene `null`. ☐

(d) L'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"ape"`, `""`. ☐

7. Considerate il seguente metodo ricorsivo. Scrivete il risultato restituito dalle chiamate indicate nei due riquadri:

```
... int f(int x) {
  if (x <= 1)
    return 3;
  else
    return x * f(x / 2) + 2;
}
```

f(3)	f(6)