Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta dell'8 settembre 2014

TEMPO DISPONIBILE: 2 ore

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Number`: ogni oggetto della classe rappresenta un numero. 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 Integer sommaInteger()`
    Restituisce il riferimento a un oggetto di tipo `Integer` che rappresenta la somma dei valori degli oggetti di tipo `Integer` presenti nell'elenco.
  - `public int quantiInteger()`
    Restituisce il numero di oggetti di tipo `Integer` presenti nell'elenco.
  - `public int quanti()`
    Restituisce il numero totale di oggetti presenti nell'elenco.
  - `public double frequenzaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla *frequenza* degli oggetti di tipo `Integer` nell'elenco, definita come il rapporto tra in numero di oggetti di tipo `Integer` e il numero totale di oggetti. Ad esempio, se l'elenco contiene `10` oggetti e `3` di questi sono di tipo `Integer` la frequenza è `0.3` (cioè `3` diviso `10`).

1. Scrivete l'implementazione del metodo `frequenzaInteger`, *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[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori aritmetici come* `+` *a riferimenti di tipo* `Integer` *(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 y;
  private static int w = 3;

  public Beta(String s, int t) {
    super(t + s.length());
    y = s.length() / 2;
    w = w * w;
  }

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

  public static int getStatico() {
    return w;
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa(7);
    System.out.println(a.intValue());       //2
    a = new Beta("elefante", 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` e una classe *astratta* `Delta` che estendono `Number`. Inoltre `Gamma` implementa un'interfaccia `In`.

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

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

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

- ☐ I campi statici appartengono alle classi
- ☐ Le classi astratte sono prive di campi
- ☐ Il tipo `String` di Java è primitivo
- ☐ Le classi astratte sono prive di metodi concreti
- ☐ Rimuovendo "`extends Alfa`" nell'intestazione della classe, il compilatore segnala almeno un errore nel codice di `Beta`
- ☐ Rimuovendo "`extends Alfa`" nell'intestazione della classe, il compilatore segnala almeno due errori nel codice di `Beta`
- ☐ Il compilatore permette l'esecuzione del bytecode
- ☐ "bytecode" è un altro nome del linguaggio Java
- ☐ In Java è possibile definire un nuovo tipo riferimento solo definendo una nuova classe
- ☐ Ogni classe (eccetto `Object`) possiede almeno un costruttore che richiama implicitamente o esplicitamente un costruttore della superclasse
- ☐ `int` è un'abbreviazione di `Integer`
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ In Java una classe può implementare direttamente più interfacce
- ☐ Le interfacce possono contenere campi non statici
- ☐ Le eccezioni di tipo `IOException` sono controllate
- ☐ 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
- ☐ Il garbage collector serve per ripulire lo stack eliminando la frammentazione
- ☐ In Java gli array contenenti valori di tipo `int` non sono oggetti

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

```
int x = 1;
try {
  do
    x = nomi[x].length();
  while (x / x == x / x);
} catch (ArithmeticException e) {
  x = x + 41;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 25;
} catch (NullPointerException e) {
  x = x + 23;
}
```

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 / 2 * 2 != x)
    return x + 1;
  else
    return x * f(x / 2) + 2;
}
```

f(3)	f(6)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta dell'8 settembre 2014

TEMPO DISPONIBILE: 2 ore

Negli esercizi proposti si utilizzano le seguenti classi:

- Classe astratta `Number`: ogni oggetto della classe rappresenta un numero. 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 Integer sommaInteger()`
    Restituisce il riferimento a un oggetto di tipo `Integer` che rappresenta la somma dei valori degli oggetti di tipo `Integer` presenti nell'elenco.
  - `public int quantiInteger()`
    Restituisce il numero di oggetti di tipo `Integer` presenti nell'elenco.
  - `public int quanti()`
    Restituisce il numero totale di oggetti presenti nell'elenco.
  - `public double frequenzaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla *frequenza* degli oggetti di tipo `Integer` nell'elenco, definita come il rapporto tra in numero di oggetti di tipo `Integer` e il numero totale di oggetti. Ad esempio, se l'elenco contiene `10` oggetti e `3` di questi sono di tipo `Integer` la frequenza è `0.3` (cioè `3` diviso `10`).

1. Scrivete l'implementazione del metodo `frequenzaInteger`, *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[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori aritmetici come* `+` *a riferimenti di tipo* `Integer` *(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 y;
  private static int w = 2;

  public Beta(String s, int t) {
    super(t + s.length());
    y = s.length() / 2;
    w = w * w;
  }

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

  public static int getStatico() {
    return w;
  }
}
```

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa(8);
    System.out.println(a.intValue());       //2
    a = new Beta("cicala", 5);
    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` e una classe *astratta* `Delta` che estendono `Number`. Inoltre `Gamma` implementa un'interfaccia `In`.

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

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

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

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

- ☐ Rimuovendo "`extends Alfa`" nell'intestazione della classe, il compilatore segnala almeno un errore nel codice di `Beta`
- ☐ I campi statici appartengono alle classi
- ☐ 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` non sono oggetti
- ☐ In Java è possibile definire un nuovo tipo riferimento solo definendo una nuova classe
- ☐ Le classi astratte sono prive di metodi concreti
- ☐ Rimuovendo "`extends Alfa`" nell'intestazione della classe, il compilatore segnala almeno due errori nel codice di `Beta`
- ☐ Il compilatore permette l'esecuzione del bytecode
- ☐ Ogni classe (eccetto `Object`) possiede almeno un costruttore che richiama implicitamente o esplicitamente un costruttore della superclasse
- ☐ 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
- ☐ Il garbage collector serve per ripulire lo stack eliminando la frammentazione
- ☐ `int` è un'abbreviazione di `Integer`
- ☐ Le classi astratte sono prive di campi
- ☐ Il tipo `String` di Java è primitivo
- ☐ Le interfacce possono contenere campi non statici
- ☐ Le eccezioni di tipo `IOException` sono controllate
- ☐ "bytecode" è un altro nome del linguaggio Java
- ☐ Gli oggetti sono memorizzati nello heap

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

```
int x = 1;
try {
  do
    x = nomi[x].length();
  while (x / x == x / x);
} catch (ArithmeticException e) {
  x = x + 38;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 28;
} catch (NullPointerException e) {
  x = x + 20;
}
```

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 / 2 * 2 != x)
    return x + 2;
  else
    return x * f(x / 2) + 1;
}
```

f(3)	f(6)