Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 16 giugno 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`.

  Tra i metodi della classe `Number` vi è:

  - `public abstract int intValue()`

- Classe `ElencoNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public int sommaInteriDispari()`
    Restituisce un valore di tipo `int` uguale alla somma degli oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.
  - `public int quantiInteriDispari()`
    Restituisce il numero di oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.
  - `public double mediaInteriDispari()`
    Restituisce un valore *di tipo* `double` uguale alla media degli oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.

1. Scrivete l'implementazione del metodo `mediaInteriDispari`, *senza conoscere* l'implementazione di `ElencoNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che la sequenza contenga sempre almeno un oggetto di tipo `Integer` che rappresenti un numero dispari.

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che il metodo `mediaInteriDispari` sollevi una eccezione di tipo `ArithmeticException` se la sequenza non contiene alcun oggetto di tipo `Integer` che rappresenti un numeri dispari (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] dati
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteriDispari`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare 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 `String`. Tra i metodi di `Alfa` vi è `public int intValue()` che restituisce la lunghezza della stringa specificata al momento della creazione dell'oggetto. Ad esempio, il metodo `intValue()` di un oggetto costruito invocando `new Alfa("pippo")` restituisce `5`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int z;
  private static int w = 2;

  public Beta(String s, int t) {
    super(s);
    z = t;
    w = w * t;
  }

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa("elefante");
    System.out.println(a.intValue());       //2
    a = new Beta("gatto", 6);
    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 `Gamma` che estende `Number` e implementa un'interfaccia `In`, e una classe `Delta` che estende `Alfa`.

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

- ☐ `Delta` deve fornire l'implementazione di tutti i metodi 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`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ `Number` è un supertipo di `Gamma`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Ogni istanza di `Beta` contiene il campo `w`
- ☐ `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

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

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

- ☐ Il programmatore in Java può definire nuovi tipi riferimento
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Durante l'esecuzione le variabili definite nel metodo `main` si trovano nello heap
- ☐ Il garbage collector recupera blocchi liberi nello stack
- ☐ Gli oggetti sono memorizzati nello stack
- ☐ I campi statici appartengono alle classi
- ☐ In un'interfaccia è possibile definire costruttori
- ☐ Il tipo `String` di Java è primitivo
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Number`, il compilatore segnala un errore
- ☐ In Java una classe può essere definita estendendo direttamente più classi
- ☐ La Java Virtual Machine permette di compilare programmi scritti in Java
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Una classe dichiarata `final` non può essere estesa
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java una variabile di tipo `Object` può memorizzare un riferimento a un valore primitivo qualsiasi
- ☐ In Java una classe può implementare più interfacce
- ☐ In Java un'interfaccia può essere definita estendendo direttamente più interfacce
- ☐ Il programmatore in Java può definire nuovi tipi primitivi

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

```
int x = 2;
try {
  do
    x = nomi[x].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 22;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 33;
} catch (NullPointerException e) {
  x = x + 11;
}
```

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 `""`, `"cane"`, `"ape"`, `"formica"`. ☐

(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 * x == x + x)
    return x;
  else
    return 2 * f(x / 2) + 3;
}
```

f(1)	f(5)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 16 giugno 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`.

  Tra i metodi della classe `Number` vi è:

  - `public abstract int intValue()`

- Classe `ElencoNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public int sommaInteriDispari()`
    Restituisce un valore di tipo `int` uguale alla somma degli oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.
  - `public int quantiInteriDispari()`
    Restituisce il numero di oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.
  - `public double mediaInteriDispari()`
    Restituisce un valore *di tipo* `double` uguale alla media degli oggetti di tipo `Integer` presenti nella sequenza che rappresentano numeri dispari.

1. Scrivete l'implementazione del metodo `mediaInteriDispari`, *senza conoscere* l'implementazione di `ElencoNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che la sequenza contenga sempre almeno un oggetto di tipo `Integer` che rappresenti un numero dispari.

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che il metodo `mediaInteriDispari` sollevi una eccezione di tipo `ArithmeticException` se la sequenza non contiene alcun oggetto di tipo `Integer` che rappresenti un numeri dispari (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] dati
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteriDispari`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare 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 `String`. Tra i metodi di `Alfa` vi è `public int intValue()` che restituisce la lunghezza della stringa specificata al momento della creazione dell'oggetto. Ad esempio, il metodo `intValue()` di un oggetto costruito invocando `new Alfa("pippo")` restituisce `5`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int z;
  private static int w = 3;

  public Beta(String s, int t) {
    super(s);
    z = t;
    w = w * t;
  }

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

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

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

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

- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Ogni istanza di `Beta` contiene il campo `w`
- ☐ `Delta` deve fornire l'implementazione di tutti i metodi 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`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ `Number` è un supertipo di `Gamma`

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

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

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

- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java una variabile di tipo `Object` può memorizzare un riferimento a un valore primitivo qualsiasi
- ☐ In Java una classe può implementare più interfacce
- ☐ In Java un'interfaccia può essere definita estendendo direttamente più interfacce
- ☐ In Java una classe può essere definita estendendo direttamente più classi
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Number`, il compilatore segnala un errore
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Durante l'esecuzione le variabili definite nel metodo `main` si trovano nello heap
- ☐ Il garbage collector recupera blocchi liberi nello stack
- ☐ Gli oggetti sono memorizzati nello stack
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ I campi statici appartengono alle classi
- ☐ In un'interfaccia è possibile definire costruttori
- ☐ Il tipo `String` di Java è primitivo
- ☐ La Java Virtual Machine permette di compilare programmi scritti in Java
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Una classe dichiarata `final` non può essere estesa
- ☐ Il programmatore in Java può definire nuovi tipi primitivi
- ☐ Il programmatore in Java può definire nuovi tipi riferimento

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

```
int x = 2;
try {
  do
    x = nomi[x].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 11;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 22;
} catch (NullPointerException e) {
  x = x + 33;
}
```

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 `""`, `"cane"`, `"ape"`, `"formica"`. ☐

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

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

f(1)	f(5)