

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 19 settembre 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

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. Tra i metodi della classe `Number` vi è:

```
public abstract double doubleValue()
```

Restituisce un valore di tipo `double` uguale al numero rappresentato dall'oggetto che esegue il metodo (dopo avere effettuato le necessarie conversioni e approssimazioni; ad esempio se l'oggetto rappresenta il numero 1, il metodo restituirà il valore 1 rappresentato come `double`).

- Classi concrete `Integer`, `Long`, `Float`, `Double`.
Gli oggetti rappresentano valori interi o in virgola mobile dei tipi primitivi `int`, `long`, `float` e `double`, rispettivamente. Queste classi (e non solo queste) sono definite estendendo direttamente `Number`.

Tra i metodi forniti dalla classe `Double` vi sono:

```
public boolean isInfinite()
```

Restituisce `true` se l'oggetto che esegue il metodo rappresenta un valore infinito.

```
public boolean isNaN()
```

Restituisce `true` se l'oggetto che esegue il metodo rappresenta il valore `NaN` (Not-a-Number).

Attenzione: nello svolgere gli esercizi fate riferimento a quanto indicato sopra e non all'implementazione delle classi, che è privata e dunque non accessibile al di fuori del codice di ciascuna delle classi stesse.

Negli esercizi 1, 2 e 3 considerate la dichiarazione di variabile `Number[] ar`. Supponete di disporre di una porzione di codice alla fine della quale `ar` si riferisca a un array in cui ogni posizione contiene il riferimento a un oggetto di tipo `Number` (quindi nessuna delle posizioni contiene `null`).

1. Scrivete una porzione di codice per calcolare e visualizzare la somma dei valori negativi rappresentati dagli oggetti contenuti nell'array `ar`. Per il calcolo utilizzate una variabile di tipo `double` (il valore di tipo primitivo `double` di ogni elemento dell'array può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero totale di oggetti contenuti nell'array `ar` che non siano di tipo `Double`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di oggetti di tipo `Double` contenuti nell'array `ar` che rappresentano il valore `NaN`.

4. Considerate le seguenti classi:

```
public class Strana {
    private static double d = 4.4;
    private int k;
    private static int w;

    public Strana() {
        this(1);
    }

    public Strana(int x) {
        k = x;
        w = w + (int) d;
        d = d + 1;
    }

    public int intValue() {
        return k + w;
    }

    public static double getD() {
        return d;
    }
}
```

```
class Prova {
    public static void main(String[] args) {
        Strana s = new Strana(3);
        System.out.println(s.intValue()); //1
        System.out.println(Strana.getD()); //2
        s = new Strana();
        System.out.println(s.intValue()); //3
        System.out.println(Strana.getD()); //4
    }
}
```

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

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

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta **Alfa** che estende **Number**, una classe concreta **Beta** che estende **Alfa** e una classe concreta **Gamma** che estende **Strana**. Le classi **Beta** e **Gamma** implementano un'interfaccia **In**.

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

- Beta deve fornire l'implementazione del metodo `doubleValue`
- Gamma deve fornire l'implementazione del metodo `doubleValue`
- Le classi dichiarate `final` non possiedono superclassi
- Alfa deve fornire l'implementazione del metodo `doubleValue`
- Ogni oggetto della classe **Strana** possiede esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- Ogni oggetto della classe **Strana** possiede esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- Ogni oggetto della classe **Strana** possiede esattamente 1 campo (oltre a quelli ereditati dalla superclasse)
- In** possiede almeno un costruttore
- Se il codice di **Gamma** non contiene un costruttore, il compilatore ne aggiunge automaticamente uno
- Se il codice di **Gamma** non contiene un costruttore privo di argomenti, il compilatore lo aggiunge in ogni caso

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

- `n = a`
- `n = i`
- `n = new Number(...)` (al posto di ... ci sono gli argomenti richiesti dal costruttore)
- `g = s`
- `s = new Strana((new Strana()).intValue())`
- `b = a`
- `n = new Beta(...)` (al posto di ... ci sono gli argomenti richiesti dal costruttore)
- `g = (Gamma) s`
- `a = b`
- `i = (Alfa) n`

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

```
int x = 0;
try {
    for (Number n: numeri)
        x = x + (int)n.doubleValue();
    x = (Integer)numeri[x] / x;
} catch (ArithmeticException e) {
    x = 42 + x;
} catch (ArrayIndexOutOfBoundsException e) {
    x = 52 + x;
} catch (NullPointerException e) {
    x = 62 + x;
} catch (ClassCastException e) {
    x = 72 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenta di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenta di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenta di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

- (a) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero -1 e il riferimento a oggetto `Double` che rappresenta il numero 1.
- (b) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero 1 e il riferimento a oggetto `Double` che rappresenta il numero 1.
- (c) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Double` che rappresenta il numero 0 e il riferimento a oggetto `Integer` che rappresenta il numero 1.
- (d) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero 0 e il riferimento a oggetto `Double` che rappresenta il numero 1.

7. Considerate il seguente metodo ricorsivo e le due chiamate indicate dei riquadri. Per ciascuna di esse, nel caso la chiamata termini correttamente scrivete il risultato restituito, altrimenti scrivete ERR:

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

f(3)	f(7)
------	------

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 19 settembre 2016

TEMPO DISPONIBILE: 1 ora e 40 minuti

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. Tra i metodi della classe `Number` vi è:

```
public abstract double doubleValue()
```

Restituisce un valore di tipo `double` uguale al numero rappresentato dall'oggetto che esegue il metodo (dopo avere effettuato le necessarie conversioni e approssimazioni; ad esempio se l'oggetto rappresenta il numero 1, il metodo restituirà il valore 1 rappresentato come `double`).

- Classi concrete `Integer`, `Long`, `Float`, `Double`.

Gli oggetti rappresentano valori interi o in virgola mobile dei tipi primitivi `int`, `long`, `float` e `double`, rispettivamente. Queste classi (e non solo queste) sono definite estendendo direttamente `Number`.

Tra i metodi forniti dalla classe `Double` vi sono:

```
public boolean isInfinite()
```

Restituisce `true` se l'oggetto che esegue il metodo rappresenta un valore infinito.

```
public boolean isNaN()
```

Restituisce `true` se l'oggetto che esegue il metodo rappresenta il valore `NaN` (Not-a-Number).

Attenzione: nello svolgere gli esercizi fate riferimento a quanto indicato sopra e non all'implementazione delle classi, che è privata e dunque non accessibile al di fuori del codice di ciascuna delle classi stesse.

Negli esercizi 1, 2 e 3 considerate la dichiarazione di variabile `Number[] ar`. Supponete di disporre di una porzione di codice alla fine della quale `ar` si riferisca a un array in cui ogni posizione contiene il riferimento a un oggetto di tipo `Number` (quindi nessuna delle posizioni contiene `null`).

1. Scrivete una porzione di codice per calcolare e visualizzare la somma dei valori negativi rappresentati dagli oggetti contenuti nell'array `ar`. Per il calcolo utilizzate una variabile di tipo `double` (il valore di tipo primitivo `double` di ogni elemento dell'array può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero totale di oggetti contenuti nell'array `ar` che non siano di tipo `Double`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di oggetti di tipo `Double` contenuti nell'array `ar` che rappresentano il valore `NaN`.

4. Considerate le seguenti classi:

```
public class Strana {
    private static double d = 3.3;
    private int k;
    private static int w;

    public Strana() {
        this(1);
    }

    public Strana(int x) {
        k = x;
        w = w + (int) d;
        d = d + 1;
    }

    public int intValue() {
        return k + w;
    }

    public static double getD() {
        return d;
    }
}
```

```
class Prova {
    public static void main(String[] args) {
        Strana s = new Strana(4);
        System.out.println(s.intValue()); //1
        System.out.println(Strana.getD()); //2
        s = new Strana();
        System.out.println(s.intValue()); //3
        System.out.println(Strana.getD()); //4
    }
}
```

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

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

5. Oltre alle classi utilizzate negli esercizi precedenti, considerate una classe astratta **Gamma** che estende **Number**, una classe concreta **Delta** che estende **Gamma** e una classe concreta **Alfa** che estende **Strana**. Le classi **Delta** e **Alfa** implementano 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 **Strana** possiede esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- Ogni oggetto della classe **Strana** possiede esattamente 1 campo (oltre a quelli ereditati dalla superclasse)
- In** possiede almeno un costruttore
- Se il codice di **Alfa** non contiene un costruttore, il compilatore ne aggiunge automaticamente uno
- Se il codice di **Alfa** non contiene un costruttore privo di argomenti, il compilatore lo aggiunge in ogni caso
- Le classi dichiarate **final** non possiedono superclassi
- Gamma** deve fornire l'implementazione del metodo **doubleValue**
- Ogni oggetto della classe **Strana** possiede esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- Delta** deve fornire l'implementazione del metodo **doubleValue**
- Alfa** deve fornire l'implementazione del metodo **doubleValue**

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

- `n = i`
- `n = new Delta(...)` (al posto di ... ci sono gli argomenti richiesti dal costruttore)
- `a = (Alfa) s`
- `g = d`
- `n = new Number(...)` (al posto di ... ci sono gli argomenti richiesti dal costruttore)
- `a = s`
- `s = new Strana((new Strana()).intValue())`
- `d = a`
- `n = g`
- `i = (Gamma) n`

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

```
int x = 0;
try {
    for (Number n: numeri)
        x = x + (int)n.doubleValue();
    x = (Integer)numeri[x] / x;
} catch (ArithmeticException e) {
    x = 22 + x;
} catch (ArrayIndexOutOfBoundsException e) {
    x = 32 + x;
} catch (NullPointerException e) {
    x = 42 + x;
} catch (ClassCastException e) {
    x = 52 + x;
}
```

Ricordando che:

- `ArithmeticException` viene sollevata in caso di anomalie nel calcolo di operazioni aritmetiche,
- `ArrayIndexOutOfBoundsException` viene sollevata quando si tenta di accedere a una posizione inesistente in un array,
- `NullPointerException` viene sollevata quando si tenta di accedere a un oggetto tramite un riferimento `null`,
- `ClassCastException` viene sollevata quando si tenta di forzare un oggetto a una sottoclasse di cui l'oggetto non è istanza,
- `""` indica la stringa vuota,

indicate nel riquadro corrispondente, in ciascuno dei seguenti casi, il valore della variabile `x` dopo l'esecuzione:

- (a) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Double` che rappresenta il numero 0 e il riferimento a oggetto `Integer` che rappresenta il numero 1.
- (b) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero 0 e il riferimento a oggetto `Double` che rappresenta il numero 1.
- (c) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero -1 e il riferimento a oggetto `Double` che rappresenta il numero 1.
- (d) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero 1 e il riferimento a oggetto `Double` che rappresenta il numero 1.

7. Considerate il seguente metodo ricorsivo e le due chiamate indicate dei riquadri. Per ciascuna di esse, nel caso la chiamata termini correttamente scrivete il risultato restituito, altrimenti scrivete ERR:

```
... int f(int x) {
    if (x / x == x)
        return x;
    else
        return 4 * f(x / 2) + 1;
}
```

f(3)	f(7)
------	------