Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 23 febbraio 2015

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 `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).

  La classe `Double` fornisce più costruttori. Tutti i costruttori prevedono almeno un argomento. Uno dei costruttori è:

  `public Double(double x)`
  Costruisce un oggetto che rappresenta il valore fornito tramite l'argomento.

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 sul monitor un valore di tipo `double` uguale alla somma dei valori di *tutti* gli oggetti contenuti nell'array `ar` (il valore di tipo `double` di ciascun oggetto può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor un valore di tipo `double` uguale alla somma dei valori dei *soli oggetti* di tipo `Double` contenuti nell'array `ar`.

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 un valore infinito.

4. Considerate le seguenti classi:

```
public class Strana {
  private double w;
  private static int k = 2;

  public Strana(double d, int z) {
    this(d + z);
    k = z;
  }

  public Strana(double k) {
    w = k;
  }

  public double doubleValue() {
    return w + (new Double(w)).doubleValue();
  }

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.getStatico());  //1
    Strana s = new Strana(5);
    System.out.println(s.doubleValue());      //2
    s = new Strana(5, 8);
    System.out.println(s.doubleValue());      //3
    System.out.println(Strana.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 utilizzate negli esercizi precedenti, considerate una classe concreta `Gamma` che estende `Number` e implementa un'interfaccia `In`, una classe concreta `Omega` che estende `Number`, una classe concreta `Delta` che estende `Double`.

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

- ☐ Ogni istanza di `Strana` contiene esattamente un campo (oltre a quelli ereditati dalla superclasse)
- ☐ Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Gamma` deve fornire l'implementazione del metodo `isNaN`
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Delta` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Delta` deve fornire l'implementazione del metodo `isNaN`
- ☐ `Delta` deve fornire l'implementazione dei metodi di `In`

b. Considerate le seguenti dichiarazioni di variabile:

```
Number n, Omega o, Gamma g, Long l, Delta d, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `n = (Gamma) i`
- ☐ `l = (Long) i`
- ☐ `g = d`
- ☐ `d = g`
- ☐ `d = (Delta) i`
- ☐ `g = (Gamma) i`
- ☐ `n = new Number()`
- ☐ `l = (Long) n`
- ☐ `o = n`
- ☐ `n = new Omega(...)`

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

```
int x = 2;
try {
  for (Integer i: numeri)
    x = (int) i.doubleValue();
  x = (int) numeri[x / x].doubleValue();
} catch (ArithmeticException e) {
  x = x + 40;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 25;
} catch (NullPointerException e) {
  x = x + 33;
}
```

Ricordando 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`,

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

(a) l'array riferito da `numeri` è vuoto. ☐

(b) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `10` e `0`. ☐

(c) `numeri` contiene `null`. ☐

(d) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `1`, `2` e `3`. ☐

7. Date due variabili `a` e `b` di tipo `double`, si considerino le seguenti condizioni:

il valore contenuto in `a` è maggiore di 10

il valore contenuto in `b` è al massimo 20

(a) Esprimete in linguaggio Java la *congiunzione* delle due condizioni.

(b) Esprimete in linguaggio Java la *negazione* dell'espressione ottenuta al punto (a), *senza utilizzare* l'operatore di negazione.

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 23 febbraio 2015

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 float floatValue()`
  Restituisce un valore di tipo `float` 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 `float`).

- Classi `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 `Float` 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).

  La classe `Float` fornisce più costruttori. Tutti i costruttori prevedono almeno un argomento. Uno dei costruttori è:

  `public Float(float x)`
  Costruisce un oggetto che rappresenta il valore fornito tramite l'argomento.

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 sul monitor un valore di tipo `float` uguale alla somma dei valori di *tutti* gli oggetti contenuti nell'array `ar` (il valore di tipo `float` di ciascun oggetto può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor un valore di tipo `float` uguale alla somma dei valori dei *soli oggetti* di tipo `Float` contenuti nell'array `ar`.

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

4. Considerate le seguenti classi:

```
public class Strana {
  private float w;
  private static int k = 4;

  public Strana(float f, int z) {
    this(f + z);
    k = z;
  }

  public Strana(float k) {
    w = k;
  }

  public float floatValue() {
    return w + (new Float(w)).floatValue();
  }

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.getStatico());  //1
    Strana s = new Strana(3);
    System.out.println(s.floatValue());       //2
    s = new Strana(3, 9);
    System.out.println(s.floatValue());       //3
    System.out.println(Strana.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 utilizzate negli esercizi precedenti, considerate una classe concreta `Alfa` che estende `Number` e implementa un'interfaccia `In`, una classe concreta `Omega` che estende `Number`, una classe concreta `Beta` che estende `Float`.

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

- ☐ Se il codice sorgente della classe `Alfa` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Se il codice sorgente della classe `Beta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Beta` deve fornire l'implementazione del metodo `floatValue`
- ☐ `Beta` deve fornire l'implementazione del metodo `isNaN`
- ☐ `Beta` deve fornire l'implementazione dei metodi di `In`
- ☐ `Alfa` deve fornire l'implementazione dei metodi di `In`
- ☐ `Alfa` deve fornire l'implementazione del metodo `floatValue`
- ☐ `Alfa` deve fornire l'implementazione del metodo `isNaN`
- ☐ Ogni istanza di `Strana` contiene esattamente un campo (oltre a quelli ereditati dalla superclasse)
- ☐ Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)

b. Considerate le seguenti dichiarazioni di variabile:

```
Number n, Omega o, Alfa a, Long l, Beta b, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `l = (Long) i`
- ☐ `n = new Number()`
- ☐ `l = (Long) n`
- ☐ `o = n`
- ☐ `a = b`
- ☐ `b = a`
- ☐ `b = (Beta) i`
- ☐ `a = (Alfa) i`
- ☐ `n = (Alfa) i`
- ☐ `n = new Omega(...)`

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

```
int x = 2;
try {
  for (Integer i: numeri)
    x = (int) i.floatValue();
  x = (int) numeri[x / x].floatValue();
} catch (ArithmeticException e) {
  x = x + 22;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 14;
} catch (NullPointerException e) {
  x = x + 28;
}
```

Ricordando 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`,

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

(a) `numeri` contiene `null`. ☐

(b) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `1`, `2` e `3`. ☐

(c) l'array riferito da `numeri` è vuoto. ☐

(d) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `10` e `0`. ☐

7. Date due variabili `b` e `c` di tipo `float`, si considerino le seguenti condizioni:

il valore contenuto in `b` non supera 8

il valore contenuto in `c` è più di 15

(a) Esprimete in linguaggio Java la *disgiunzione* delle due condizioni.

(b) Esprimete in linguaggio Java la *negazione* dell'espressione ottenuta al punto (a), *senza utilizzare* l'operatore di negazione.

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 23 febbraio 2015

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 `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).

  La classe `Double` fornisce più costruttori. Tutti i costruttori prevedono almeno un argomento. Uno dei costruttori è:

  `public Double(double x)`
  Costruisce un oggetto che rappresenta il valore fornito tramite l'argomento.

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[] el`. Supponete di disporre di una porzione di codice alla fine della quale `el` 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 sul monitor un valore di tipo `double` uguale alla somma dei valori di *tutti* gli oggetti contenuti nell'array `el` (il valore di tipo `double` di ciascun oggetto può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor un valore di tipo `double` uguale alla somma dei valori dei *soli oggetti* di tipo `Double` contenuti nell'array `el`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di oggetti di tipo `Double` contenuti nell'array `el` che rappresentano il valore `NaN` (Not-a-Number).

4. Considerate le seguenti classi:

```
public class Strana {
  private double w;
  private static int k = 6;

  public Strana(double d, int z) {
    this(d + z);
    k = z;
  }

  public Strana(double k) {
    w = k;
  }

  public double doubleValue() {
    return w + (new Double(w)).doubleValue();
  }

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.getStatico());  //1
    Strana s = new Strana(4);
    System.out.println(s.doubleValue());      //2
    s = new Strana(4, 3);
    System.out.println(s.doubleValue());      //3
    System.out.println(Strana.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 utilizzate negli esercizi precedenti, considerate una classe concreta `Delta` che estende `Number` e implementa un'interfaccia `In`, una classe concreta `Omega` che estende `Number`, una classe concreta `Alfa` che estende `Double`.

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

- ☐ Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Alfa` deve fornire l'implementazione dei metodi di `In`
- ☐ `Delta` deve fornire l'implementazione dei metodi di `In`
- ☐ Ogni istanza di `Strana` contiene esattamente un campo (oltre a quelli ereditati dalla superclasse)
- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Se il codice sorgente della classe `Alfa` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Alfa` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Alfa` deve fornire l'implementazione del metodo `isNaN`
- ☐ `Delta` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Delta` deve fornire l'implementazione del metodo `isNaN`

b. Considerate le seguenti dichiarazioni di variabile:

```
Number n, Omega o, Delta d, Long l, Alfa a, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `o = n`
- ☐ `n = new Omega(...)`
- ☐ `a = (Alfa) i`
- ☐ `d = (Delta) i`
- ☐ `n = new Number()`
- ☐ `n = (Delta) i`
- ☐ `l = (Long) i`
- ☐ `d = a`
- ☐ `a = d`
- ☐ `l = (Long) n`

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

```
int x = 2;
try {
  for (Integer i: numeri)
    x = (int) i.doubleValue();
  x = (int) numeri[x / x].doubleValue();
} catch (ArithmeticException e) {
  x = x + 44;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 35;
} catch (NullPointerException e) {
  x = x + 18;
}
```

Ricordando 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`,

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) riferimenti a oggetti che rappresentano gli interi `10` e `0`. ☐

(b) l'array riferito da `numeri` è vuoto. ☐

(c) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `1`, `2` e `3`. ☐

(d) `numeri` contiene `null`. ☐

7. Date due variabili `c` e `b` di tipo `double`, si considerino le seguenti condizioni:

il valore contenuto in `c` è almeno 15

il valore contenuto in `b` è minore di 23

(a) Esprimete in linguaggio Java la *congiunzione* delle due condizioni.

(b) Esprimete in linguaggio Java la *negazione* dell'espressione ottenuta al punto (a), *senza utilizzare* l'operatore di negazione.

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 23 febbraio 2015

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 float floatValue()`
  Restituisce un valore di tipo `float` 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 `float`).

- Classi `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 `Float` 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).

  La classe `Float` fornisce più costruttori. Tutti i costruttori prevedono almeno un argomento. Uno dei costruttori è:

  `public Float(float x)`
  Costruisce un oggetto che rappresenta il valore fornito tramite l'argomento.

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[] el`. Supponete di disporre di una porzione di codice alla fine della quale `el` 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 sul monitor un valore di tipo `float` uguale alla somma dei valori di *tutti* gli oggetti contenuti nell'array `el` (il valore di tipo `float` di ciascun oggetto può essere ottenuto servendosi del metodo opportuno tra quelli indicati sopra).

2. Scrivete una porzione di codice per calcolare e visualizzare sul monitor un valore di tipo `float` uguale alla somma dei valori dei *soli oggetti* di tipo `Float` contenuti nell'array `el`.

3. Scrivete una porzione di codice per calcolare e visualizzare sul monitor il numero di oggetti di tipo `Float` contenuti nell'array `el` che rappresentano il valore `NaN` (Not-a-Number).

4. Considerate le seguenti classi:

```
public class Strana {
  private float w;
  private static int k = 8;

  public Strana(float f, int z) {
    this(f + z);
    k = z;
  }

  public Strana(float k) {
    w = k;
  }

  public float floatValue() {
    return w + (new Float(w)).floatValue();
  }

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Strana.getStatico());  //1
    Strana s = new Strana(2);
    System.out.println(s.floatValue());       //2
    s = new Strana(2, 7);
    System.out.println(s.floatValue());       //3
    System.out.println(Strana.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 utilizzate negli esercizi precedenti, considerate una classe concreta `Beta` che estende `Number` e implementa un'interfaccia `In`, una classe concreta `Omega` che estende `Number`, una classe concreta `Gamma` che estende `Float`.

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 `isNaN`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ Ogni istanza di `Strana` contiene esattamente un campo (oltre a quelli ereditati dalla superclasse)
- ☐ Ogni istanza di `Strana` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ Se il codice sorgente della classe `Beta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` deve fornire l'implementazione del metodo `floatValue`
- ☐ `Beta` deve fornire l'implementazione dei metodi di `In`
- ☐ `Beta` deve fornire l'implementazione del metodo `floatValue`
- ☐ `Beta` deve fornire l'implementazione del metodo `isNaN`

b. Considerate le seguenti dichiarazioni di variabile:

```
Number n, Omega o, Beta b, Long l, Gamma g, In i;
```

Nel riquadro accanto a ciascun assegnamento scrivete SI se l'assegnamento è compilato correttamente, NO se non è compilato correttamente (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `n = new Number()`
- ☐ `g = (Gamma) i`
- ☐ `b = (Beta) i`
- ☐ `l = (Long) n`
- ☐ `l = (Long) i`
- ☐ `b = g`
- ☐ `g = b`
- ☐ `o = n`
- ☐ `n = new Omega(...)`
- ☐ `n = (Beta) i`

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

```
int x = 2;
try {
  for (Integer i: numeri)
    x = (int) i.floatValue();
  x = (int) numeri[x / x].floatValue();
} catch (ArithmeticException e) {
  x = x + 33;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 15;
} catch (NullPointerException e) {
  x = x + 27;
}
```

Ricordando 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`,

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) riferimenti a oggetti che rappresentano gli interi `1`, `2` e `3`. ☐

(b) `numeri` contiene `null`. ☐

(c) l'array riferito da `numeri` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano gli interi `10` e `0`. ☐

(d) l'array riferito da `numeri` è vuoto. ☐

7. Date due variabili `a` e `d` di tipo `float`, si considerino le seguenti condizioni:

il valore contenuto in `a` è minore di 6

il valore contenuto in `d` è almeno 18

(a) Esprimete in linguaggio Java la *disgiunzione* delle due condizioni.

(b) Esprimete in linguaggio Java la *negazione* dell'espressione ottenuta al punto (a), *senza utilizzare* l'operatore di negazione.