Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 5 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[] a`. Supponete di disporre di una porzione di codice alla fine della quale `a` 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 contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `a` che rappresentano valori negativi (per stabilire se un oggetto rappresenta un valore negativo, consideratene il valore di tipo primitivo `double`, che può essere ottenuto utilizzando il metodo opportuno tra quelli indicati sopra).

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

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

4. Considerate le seguenti classi:

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

  public Strana() {
    this(1);
  }

  public Strana(int x) {
    k = x;
    w = x + (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();
    System.out.println(s.intValue());   //1
    System.out.println(Strana.getD());  //2
    s = new Strana(3);
    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 due classi `Alfa` e `Beta` che estendono `Number`. La classe `Alfa` è astratta mentre `Beta` è concreta. Considerate inoltre una classe concreta `Gamma` che estende `Strana` e implementa un'interfaccia `In`.

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

- ☐ Uno stesso metodo può possedere differenti segnature
- ☐ In una classe ci possono essere metodi con lo stesso nome e differenti segnature
- ☐ Ogni istanza di sottoclassi di `Gamma` possiede i metodi di `In`
- ☐ `Alfa` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Beta` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Gamma` deve fornire l'implementazione del metodo `doubleValue`
- ☐ Una classe eredita tutti i metodi della superclasse, eccetto quelli dichiarati `final`
- ☐ `Alfa` non possiede costruttori
- ☐ `Gamma` eredita i costruttori di `Strana`
- ☐ `Alfa` deve contenere almeno un metodo astratto

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:

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

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 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`,
- `ClassCastException` viene sollevata quando si tenti 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 `3` e il riferimento a oggetto `Integer` che rappresenta il numero `2`.

(b) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero `3` e il riferimento a oggetto `Double` che rappresenta il numero `2`.

(c) `numeri` contiene `null`.

(d) L'array riferito da `numeri` è vuoto, cioè non contiene alcun elemento.

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 / 2 == x / 2.0)
    return 2 * f(x / 2) + 3;
  else
    return 1;
}
```

`f(4)`	`f(6)`

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 5 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[] v`. Supponete di disporre di una porzione di codice alla fine della quale `v` 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 contare e visualizzare sul monitor il numero di oggetti contenuti nell'array `v` che rappresentano valori negativi (per stabilire se un oggetto rappresenta un valore negativo, consideratene il valore di tipo primitivo `double`, che può essere ottenuto utilizzando il metodo opportuno tra quelli indicati sopra).

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

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

4. Considerate le seguenti classi:

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

  public Strana() {
    this(1);
  }

  public Strana(int x) {
    k = x;
    w = x + (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();
    System.out.println(s.intValue());   //1
    System.out.println(Strana.getD());  //2
    s = new Strana(4);
    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 due classi `Gamma` e `Delta` che estendono `Number`. La classe `Gamma` è astratta mentre `Delta` è concreta. Considerate inoltre una classe concreta `Alfa` che estende `Strana` e implementa un'interfaccia `In`.

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

- ☐ `Alfa` deve fornire l'implementazione del metodo `doubleValue`
- ☐ Una classe eredita tutti i metodi della superclasse, eccetto quelli dichiarati `final`
- ☐ `Gamma` non possiede costruttori
- ☐ `Alfa` eredita i costruttori di `Strana`
- ☐ `Gamma` deve contenere almeno un metodo astratto
- ☐ Ogni istanza di sottoclassi di `Alfa` possiede i metodi di `In`
- ☐ `Gamma` deve fornire l'implementazione del metodo `doubleValue`
- ☐ `Delta` deve fornire l'implementazione del metodo `doubleValue`
- ☐ Uno stesso metodo può possedere differenti segnature
- ☐ In una classe ci possono essere metodi con lo stesso nome e differenti segnature

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

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 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`,
- `ClassCastException` viene sollevata quando si tenti 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) `numeri` contiene `null`. ☐

(b) L'array riferito da `numeri` è vuoto, cioè non contiene alcun elemento. ☐

(c) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Double` che rappresenta il numero `3` e il riferimento a oggetto `Integer` che rappresenta il numero `2`. ☐

(d) L'array riferito da `numeri` contiene (nell'ordine indicato) il riferimento a un oggetto `Integer` che rappresenta il numero `3` e il riferimento a oggetto `Double` che rappresenta il numero `2`. ☐

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 / 2 == x / 2.0)
    return 3 * f(x / 2) + 2;
  else
    return 1;
}
```

`f(4)`	`f(6)`