Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 4 febbraio 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 `InsiemeNumeri`: ogni oggetto della classe rappresenta un insieme di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public Integer maxInteger()`
    Restituisce il riferimento all'oggetto di tipo `Integer` che rappresenta il valore più grande tra tutti gli oggetti `Integer` presenti nell'insieme. Se l'insieme non contiene nessun oggetto `Integer` il metodo restituisce `null`.
  - `public Integer minInteger()`
    Analogo al precedente, per determinare il minimo `Integer` nell'insieme.
  - `public Integer diffMaxMinInteger()`
    Restituisce la differenza tra il massimo e il minimo valore `Integer` presenti nell'insieme.

1. Scrivete l'implementazione del metodo `diffMaxMinInteger`, *senza conoscere* l'implementazione di `InsiemeNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'insieme contenga sempre almeno un valore di tipo `Integer`.

2. Riscrivete quanto richiesto per l'esercizio precedente, in modo che il metodo `diffMaxMinInteger` sollevi una eccezione di tipo `RuntimeException` se l'insieme non contiene alcun valore di tipo `Integer` (la classe `RuntimeException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'insieme. Scrivete l'implementazione del metodo `maxInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori di confronto come* `<` *e* `>` *a due riferimenti di tipo* `Integer` *(non* `Number`*!) per confrontare i valori degli oggetti associati. In alternativa, il confronto può essere effettuato utilizzando il metodo* `compareTo` *(il metodo è fornito da* `Integer`*, non da* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un *unico costruttore* che riceve come argomento un valore 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(123)` restituisce `123`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 2;

  public Beta(int s, int t) {
    super(z);
    x = s;
    y = t;
    z = s + t - z;
  }

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(5, 8);
    System.out.println(a.intValue());       //2
    a = new Alfa(10);
    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 utilizzate negli esercizi precedenti, considerate una classe `Gamma` che estende `Alfa` e implementa un'interfaccia `In`.

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

- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Number` è un supertipo di `Gamma`
- ☐ `Gamma` deve ridefinire il metodo `intValue`
- ☐ È possibile definire una sottoclasse astratta di `Number`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Alfa` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ `Beta` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` può fornire l'implementazione dei metodi di `In`

b. Considerate le seguenti dichiarazioni di variabile:

```
Integer w, Number n, 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 (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `g = n`
- ☐ `n = g`
- ☐ `n = w`
- ☐ `n = new Number(...)`
- ☐ `g = (Gamma) n`
- ☐ `a = (Gamma) i`
- ☐ `w = (Integer) n`
- ☐ `i = (Gamma) a`
- ☐ `a = (Gamma) b`
- ☐ `b = a`

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

- ☐ Tutte le eccezioni di tipo `RuntimeException` sono controllate
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ I campi statici, come `z` della classe `Alfa`, sono memorizzati nello heap
- ☐ I parametri `s` e `t` del costruttore di `Beta` vengono memorizzati nello heap
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso metodo
- ☐ Gli oggetti sono sempre memorizzati nello heap
- ☐ Tutte le eccezioni di tipo `IOException` sono controllate
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello stack
- ☐ Il tipo `int[]` è primitivo
- ☐ L'overloading dei metodi viene risolto in fase di esecuzione
- ☐ Il linguaggio Java viene chiamato anche *bytecode*
- ☐ Durante l'esecuzione lo stack contiene il codice dei costruttori e dei metodi
- ☐ In Java è possibile definire nuovi tipi riferimento
- ☐ In una stessa classe è possibile definire più metodi con la stessa segnatura, ma tipo restituito differente
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java gli array sono oggetti
- ☐ In Java un'interfaccia può possedere un costruttore
- ☐ La Java Virtual Machine è un compilatore
- ☐ La Java Virtual Machine è un interprete

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

```
int x = 0;
try {
  x = nomi[x].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 15;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 22;
} catch (NullPointerException e) {
  x = x + 29;
}
```

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`,
- `""` 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 `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"cane"`, `""`. ☐

(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` è vuoto. ☐

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

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

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 4 febbraio 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 `InsiemeNumeri`: ogni oggetto della classe rappresenta un insieme di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public Long maxLong()`
    Restituisce il riferimento all'oggetto di tipo `Long` che rappresenta il valore più grande tra tutti gli oggetti `Long` presenti nell'insieme. Se l'insieme non contiene nessun oggetto `Long` il metodo restituisce `null`.
  - `public Long minLong()`
    Analogo al precedente, per determinare il minimo `Long` nell'insieme.
  - `public Long diffMaxMinLong()`
    Restituisce la differenza tra il massimo e il minimo valore `Long` presenti nell'insieme.

1. Scrivete l'implementazione del metodo `diffMaxMinLong`, *senza conoscere* l'implementazione di `InsiemeNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'insieme contenga sempre almeno un valore di tipo `Long`.

2. Riscrivete quanto richiesto per l'esercizio precedente, in modo che il metodo `diffMaxMinLong` sollevi una eccezione di tipo `RuntimeException` se l'insieme non contiene alcun valore di tipo `Long` (la classe `RuntimeException` fornisce un costruttore che riceve come argomento una stringa).

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

`private Number[] numeri`

che si riferisce ad un array contenente i numeri presenti nell'insieme. Scrivete l'implementazione del metodo `maxLong`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori di confronto come* `<` *e* `>` *a due riferimenti di tipo* `Long` *(non* `Number`*!) per confrontare i valori degli oggetti associati. In alternativa, il confronto può essere effettuato utilizzando il metodo* `compareTo` *(il metodo è fornito da* `Long`*, non da* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un *unico costruttore* che riceve come argomento un valore 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(123)` restituisce `123`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 3;

  public Beta(int s, int t) {
    super(z);
    x = s;
    y = t;
    z = s + t - z;
  }

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(6, 9);
    System.out.println(a.intValue());       //2
    a = new Alfa(7);
    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 utilizzate negli esercizi precedenti, considerate una classe `Gamma` che estende `Alfa` 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 dei metodi astratti di `Number`
- ☐ `Beta` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Number` è un supertipo di `Gamma`
- ☐ `Gamma` deve ridefinire il metodo `intValue`
- ☐ È possibile definire una sottoclasse astratta di `Number`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` può fornire l'implementazione dei metodi di `In`
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)

b. Considerate le seguenti dichiarazioni di variabile:

```
Long w, Number n, 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 (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `a = (Gamma) b`
- ☐ `b = a`
- ☐ `g = n`
- ☐ `n = new Number(...)`
- ☐ `g = (Gamma) n`
- ☐ `a = (Gamma) i`
- ☐ `w = (Long) n`
- ☐ `i = (Gamma) a`
- ☐ `n = g`
- ☐ `n = w`

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

- ☐ Gli oggetti sono sempre memorizzati nello heap
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello stack
- ☐ Il tipo `int[]` è primitivo
- ☐ I parametri `s` e `t` del costruttore di `Beta` vengono memorizzati nello heap
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso metodo
- ☐ Il linguaggio Java viene chiamato anche *bytecode*
- ☐ Durante l'esecuzione lo stack contiene il codice dei costruttori e dei metodi
- ☐ In Java un'interfaccia può possedere un costruttore
- ☐ La Java Virtual Machine è un compilatore
- ☐ La Java Virtual Machine è un interprete
- ☐ L'overloading dei metodi viene risolto in fase di esecuzione
- ☐ In Java è possibile definire nuovi tipi riferimento
- ☐ In una stessa classe è possibile definire più metodi con la stessa segnatura, ma tipo restituito differente
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java gli array sono oggetti
- ☐ Tutte le eccezioni di tipo `RuntimeException` sono controllate
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ I campi statici, come `z` della classe `Alfa`, sono memorizzati nello heap
- ☐ Tutte le eccezioni di tipo `IOException` sono controllate

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

```
int x = 0;
try {
  x = nomi[x].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 10;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 45;
} 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`,
- `""` 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 `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `""`, `"formica"`, `"cane"`. ☐

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

(c) l'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"cane"`, `""`. ☐

(d) `nomi` contiene `null`. ☐

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

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

f(2)	f(5)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 4 febbraio 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 `InsiemeNumeri`: ogni oggetto della classe rappresenta un insieme di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public Float maxFloat()`
    Restituisce il riferimento all'oggetto di tipo `Float` che rappresenta il valore più grande tra tutti gli oggetti `Float` presenti nell'insieme. Se l'insieme non contiene nessun oggetto `Float` il metodo restituisce `null`.
  - `public Float minFloat()`
    Analogo al precedente, per determinare il minimo `Float` nell'insieme.
  - `public Float diffMaxMinFloat()`
    Restituisce la differenza tra il massimo e il minimo valore `Float` presenti nell'insieme.

1. Scrivete l'implementazione del metodo `diffMaxMinFloat`, *senza conoscere* l'implementazione di `InsiemeNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'insieme contenga sempre almeno un valore di tipo `Float`.

2. Riscrivete quanto richiesto per l'esercizio precedente, in modo che il metodo `diffMaxMinFloat` sollevi una eccezione di tipo `RuntimeException` se l'insieme non contiene alcun valore di tipo `Float` (la classe `RuntimeException` fornisce un costruttore che riceve come argomento una stringa).

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

`private Number[] numeri`

che si riferisce ad un array contenente i numeri presenti nell'insieme. Scrivete l'implementazione del metodo `maxFloat`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori di confronto come* `<` *e* `>` *a due riferimenti di tipo* `Float` *(non* `Number`*!) per confrontare i valori degli oggetti associati. In alternativa, il confronto può essere effettuato utilizzando il metodo* `compareTo` *(il metodo è fornito da* `Float`*, non da* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un *unico costruttore* che riceve come argomento un valore 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(123)` restituisce `123`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 4;

  public Beta(int s, int t) {
    super(z);
    x = s;
    y = t;
    z = s + t - z;
  }

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(5, 10);
    System.out.println(a.intValue());       //2
    a = new Alfa(18);
    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 utilizzate negli esercizi precedenti, considerate una classe `Gamma` che estende `Alfa` e implementa un'interfaccia `In`.

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

- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` può fornire l'implementazione dei metodi di `In`
- ☐ `Number` è un supertipo di `Gamma`
- ☐ `Gamma` deve ridefinire il metodo `intValue`
- ☐ È possibile definire una sottoclasse astratta di `Number`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Beta` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Alfa` deve fornire l'implementazione dei metodi astratti di `Number`

b. Considerate le seguenti dichiarazioni di variabile:

```
Float w, Number n, 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 (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `w = (Float) n`
- ☐ `i = (Gamma) a`
- ☐ `n = g`
- ☐ `n = w`
- ☐ `g = n`
- ☐ `n = new Number(...)`
- ☐ `b = a`
- ☐ `g = (Gamma) n`
- ☐ `a = (Gamma) i`
- ☐ `a = (Gamma) b`

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

- ☐ Il linguaggio Java viene chiamato anche *bytecode*
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java gli array sono oggetti
- ☐ In Java un'interfaccia può possedere un costruttore
- ☐ La Java Virtual Machine è un compilatore
- ☐ La Java Virtual Machine è un interprete
- ☐ L'overloading dei metodi viene risolto in fase di esecuzione
- ☐ I campi statici, come `z` della classe `Alfa`, sono memorizzati nello heap
- ☐ I parametri `s` e `t` del costruttore di `Beta` vengono memorizzati nello heap
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso metodo
- ☐ Gli oggetti sono sempre memorizzati nello heap
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ Tutte le eccezioni di tipo `IOException` sono controllate
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello stack
- ☐ Il tipo `int[]` è primitivo
- ☐ Durante l'esecuzione lo stack contiene il codice dei costruttori e dei metodi
- ☐ In Java è possibile definire nuovi tipi riferimento
- ☐ In una stessa classe è possibile definire più metodi con la stessa segnatura, ma tipo restituito differente
- ☐ Tutte le eccezioni di tipo `RuntimeException` sono controllate
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap

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

```
int x = 0;
try {
  x = nomi[x].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 4;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 8;
} catch (NullPointerException e) {
  x = x + 12;
}
```

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`,
- `""` 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 `nomi` è vuoto.

(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"`, `"cane"`, `""`.

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

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

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 4 febbraio 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 `InsiemeNumeri`: ogni oggetto della classe rappresenta un insieme di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public Double maxDouble()`
    Restituisce il riferimento all'oggetto di tipo `Double` che rappresenta il valore più grande tra tutti gli oggetti `Double` presenti nell'insieme. Se l'insieme non contiene nessun oggetto `Double` il metodo restituisce `null`.
  - `public Double minDouble()`
    Analogo al precedente, per determinare il minimo `Double` nell'insieme.
  - `public Double diffMaxMinDouble()`
    Restituisce la differenza tra il massimo e il minimo valore `Double` presenti nell'insieme.

1. Scrivete l'implementazione del metodo `diffMaxMinDouble`, *senza conoscere* l'implementazione di `InsiemeNumeri`, ma utilizzando gli altri metodi forniti dalla classe. In questa versione supponete che l'insieme contenga sempre almeno un valore di tipo `Double`.

2. Riscrivete quanto richiesto per l'esercizio precedente, in modo che il metodo `diffMaxMinDouble` sollevi una eccezione di tipo `RuntimeException` se l'insieme non contiene alcun valore di tipo `Double` (la classe `RuntimeException` fornisce un costruttore che riceve come argomento una stringa).

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

`private Number[] numeri`

che si riferisce ad un array contenente i numeri presenti nell'insieme. Scrivete l'implementazione del metodo `maxDouble`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori di confronto come* `<` *e* `>` *a due riferimenti di tipo* `Double` *(non* `Number`*!) per confrontare i valori degli oggetti associati. In alternativa, il confronto può essere effettuato utilizzando il metodo* `compareTo` *(il metodo è fornito da* `Double`*, non da* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un *unico costruttore* che riceve come argomento un valore 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(123)` restituisce `123`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 5;

  public Beta(int s, int t) {
    super(z);
    x = s;
    y = t;
    z = s + t - z;
  }

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Beta(3, 8);
    System.out.println(a.intValue());       //2
    a = new Alfa(11);
    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 utilizzate negli esercizi precedenti, considerate una classe `Gamma` che estende `Alfa` e implementa un'interfaccia `In`.

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

- ☐ Ogni istanza di `Beta` contiene esattamente 2 campi (oltre a quelli ereditati dalla superclasse)
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Alfa` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ `Beta` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ `Number` è un supertipo di `Gamma`
- ☐ È possibile definire una sottoclasse astratta di `Number`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` può fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` deve ridefinire il metodo `intValue`

b. Considerate le seguenti dichiarazioni di variabile:

```
Double w, Number n, 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 (supponete che al posto di `...` vi siano gli argomenti opportuni):

- ☐ `g = (Gamma) n`
- ☐ `n = new Number(...)`
- ☐ `w = (Double) n`
- ☐ `i = (Gamma) a`
- ☐ `g = n`
- ☐ `b = a`
- ☐ `n = g`
- ☐ `n = w`
- ☐ `a = (Gamma) i`
- ☐ `a = (Gamma) b`

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

- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso metodo
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello stack
- ☐ In Java un'interfaccia può possedere un costruttore
- ☐ La Java Virtual Machine è un compilatore
- ☐ La Java Virtual Machine è un interprete
- ☐ Tutte le eccezioni di tipo `RuntimeException` sono controllate
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ In Java è possibile definire nuovi tipi primitivi
- ☐ L'overloading dei metodi viene risolto in fase di esecuzione
- ☐ Il linguaggio Java viene chiamato anche *bytecode*
- ☐ Durante l'esecuzione lo stack contiene il codice dei costruttori e dei metodi
- ☐ In Java è possibile definire nuovi tipi riferimento
- ☐ In una stessa classe è possibile definire più metodi con la stessa segnatura, ma tipo restituito differente
- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ In Java gli array sono oggetti
- ☐ I campi statici, come `z` della classe `Alfa`, sono memorizzati nello heap
- ☐ Il tipo `int[]` è primitivo
- ☐ I parametri `s` e `t` del costruttore di `Beta` vengono memorizzati nello heap
- ☐ Gli oggetti sono sempre memorizzati nello heap
- ☐ Tutte le eccezioni di tipo `IOException` sono controllate

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

```
int x = 0;
try {
  x = nomi[x].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 26;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 12;
} catch (NullPointerException e) {
  x = x + 19;
}
```

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`,
- `""` 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 `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `""`, `"formica"`, `"cane"`. ☐

(b) `nomi` contiene `null`. ☐

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

(d) l'array riferito da `nomi` contiene (nell'ordine indicato) riferimenti a oggetti che rappresentano le stringhe `"formica"`, `"cane"`, `""`. ☐

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

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

f(2)	f(5)