Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 11 luglio 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 un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public Integer interoMax()`
    `public Integer interoMin()`
    Restituiscono il riferimento a un oggetto di tipo `Integer` che rappresenta, rispettivamente, il più grande e il più piccolo valore di tipo `Integer` presente nell'elenco. Se l'elenco non contiene alcun oggetto di tipo `Integer` entrambi i metodi restituiscono il riferimento `null`.
  - `public double mediaInteriMinMax()`
    Restituisce un valore *di tipo* `double` uguale alla media tra il più grande e il più piccolo valore di tipo `Integer` presenti nell'elenco. Ad esempio, se il valore più grande è `10` e il più piccolo è `3`, il metodo restituisce `6.5`.

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

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che nel caso l'elenco non contenga alcun oggetto di tipo `Integer` il metodo `mediaInteriMinMax` sollevi una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `interoMin`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori di confronto come* `<` *e* `>` *a 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`, 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 y;
  private static int w = 3, z = 4;

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

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa("cane");
    System.out.println(a.intValue());       //2
    a = new Beta("formica", 8);
    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 *astratta* `Gamma` che estende `Number` e una classe concreta `Delta` 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:

- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ `Gamma` è il nome di un tipo riferimento
- ☐ `Object` è un supertipo di `Beta`
- ☐ Ogni istanza della classe `Delta` contiene il campo `w`
- ☐ `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
- ☐ `Delta` deve fornire l'implementazione dei metodi di `In`
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)

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

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

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

- ☐ Durante l'esecuzione, nello stack si può avere frammentazione
- ☐ In Java gli array sono oggetti
- ☐ 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*
- ☐ Se il codice sorgente di una classe non contiene un costruttore privo di argomenti allora il compilatore lo aggiunge automaticamente
- ☐ Il linguaggio Java viene anche chiamato bytecode
- ☐ 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
- ☐ La lazy evaluation permette di eseguire in maniera più efficiente le operazioni aritmetiche tra numeri
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ I campi statici appartengono alle classi
- ☐ Le classi astratte sono prive di costruttori
- ☐ Il tipo `Object` 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
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Le eccezioni di tipo `ArithmeticException` sono controllate
- ☐ L'overriding viene risolto dal compilatore
- ☐ 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 oggetto qualsiasi

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

```
int x = 2;
try {
  do
    x = nomi[x - 1].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 15;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 29;
} catch (NullPointerException e) {
  x = x + 17;
}
```

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 % 2 == 1 || f(x / 2) < 3)
    return x;
  else
    return x - 1;
}
```

f(2)	f(8)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 11 luglio 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 un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public Integer interoMax()`
    `public Integer interoMin()`
    Restituiscono il riferimento a un oggetto di tipo `Integer` che rappresenta, rispettivamente, il più grande e il più piccolo valore di tipo `Integer` presente nell'elenco. Se l'elenco non contiene alcun oggetto di tipo `Integer` entrambi i metodi restituiscono il riferimento `null`.
  - `public double mediaInteriMinMax()`
    Restituisce un valore *di tipo* `double` uguale alla media tra il più grande e il più piccolo valore di tipo `Integer` presenti nell'elenco. Ad esempio, se il valore più grande è `10` e il più piccolo è `3`, il metodo restituisce `6.5`.

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

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che nel caso l'elenco non contenga alcun oggetto di tipo `Integer` il metodo `mediaInteriMinMax` sollevi una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `interoMin`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori di confronto come* `<` *e* `>` *a 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`, 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 y;
  private static int w = 3, z = 2;

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

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa("formica");
    System.out.println(a.intValue());       //2
    a = new Beta("cane", 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 *astratta* `Gamma` che estende `Number` e una classe concreta `Delta` 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:

- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` è il nome di un tipo riferimento
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ `Object` è un supertipo di `Beta`
- ☐ Ogni istanza della classe `Delta` contiene il campo `w`
- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)
- ☐ `Delta` deve 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: `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

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

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

- ☐ In Java una classe può essere definita estendendo direttamente più classi
- ☐ L'overriding viene risolto dal compilatore
- ☐ In Java gli array sono oggetti
- ☐ Il programmatore in Java può definire nuovi tipi riferimento
- ☐ 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*
- ☐ Se il codice sorgente di una classe non contiene un costruttore privo di argomenti allora il compilatore lo aggiunge automaticamente
- ☐ Il linguaggio Java viene anche chiamato bytecode
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ I campi statici appartengono alle classi
- ☐ 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 oggetto qualsiasi
- ☐ Durante l'esecuzione, nello stack si può avere frammentazione
- ☐ La lazy evaluation permette di eseguire in maniera più efficiente le operazioni aritmetiche tra numeri
- ☐ Le classi astratte sono prive di costruttori
- ☐ Il tipo `Object` di Java è primitivo
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Le eccezioni di tipo `ArithmeticException` sono controllate

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

```
int x = 2;
try {
  do
    x = nomi[x - 1].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 19;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 23;
} catch (NullPointerException e) {
  x = x + 12;
}
```

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 % 2 == 1 || f(x / 2) < 3)
    return x;
  else
    return x + 1;
}
```

f(2)	f(8)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 11 luglio 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 un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public Integer interoMax()`
    `public Integer interoMin()`
    Restituiscono il riferimento a un oggetto di tipo `Integer` che rappresenta, rispettivamente, il più grande e il più piccolo valore di tipo `Integer` presente nell'elenco. Se l'elenco non contiene alcun oggetto di tipo `Integer` entrambi i metodi restituiscono il riferimento `null`.
  - `public double mediaInteriMinMax()`
    Restituisce un valore *di tipo* `double` uguale alla media tra il più grande e il più piccolo valore di tipo `Integer` presenti nell'elenco. Ad esempio, se il valore più grande è `10` e il più piccolo è `3`, il metodo restituisce `6.5`.

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

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che nel caso l'elenco non contenga alcun oggetto di tipo `Integer` il metodo `mediaInteriMinMax` sollevi una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `interoMin`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori di confronto come* `<` *e* `>` *a 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`, 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 y;
  private static int w = 2, z = 6;

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

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

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

```
class Prova {
  public static void main(String[] args) {
    System.out.println(Beta.getStatico());  //1
    Alfa a = new Alfa("balena");
    System.out.println(a.intValue());       //2
    a = new Beta("elefante", 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 degli esercizi precedenti, considerate una classe *astratta* `Gamma` che estende `Number` e una classe concreta `Delta` 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:

- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)
- ☐ `Delta` deve fornire l'implementazione dei metodi di `In`
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` è il nome di un tipo riferimento
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ `Object` è un supertipo di `Beta`
- ☐ Ogni istanza della classe `Delta` contiene il campo `w`

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 = n`
- ☐ `i = d`
- ☐ `a = (Beta) n`
- ☐ `g = d`
- ☐ `i = (Gamma) n`
- ☐ `g = (Beta) a`
- ☐ `n = b`
- ☐ `n = (Beta) a`
- ☐ `d = (In) 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 oggetto qualsiasi
- ☐ Durante l'esecuzione, nello stack si può avere frammentazione
- ☐ La lazy evaluation permette di eseguire in maniera più efficiente le operazioni aritmetiche tra numeri
- ☐ 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*
- ☐ Se il codice sorgente di una classe non contiene un costruttore privo di argomenti allora il compilatore lo aggiunge automaticamente
- ☐ Il linguaggio Java viene anche chiamato bytecode
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ I campi statici appartengono alle classi
- ☐ Le classi astratte sono prive di costruttori
- ☐ Il tipo `Object` di Java è primitivo
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Le eccezioni di tipo `ArithmeticException` sono controllate
- ☐ L'overriding viene risolto dal compilatore
- ☐ In Java gli array sono oggetti
- ☐ 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 - 1].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 27;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 16;
} catch (NullPointerException e) {
  x = x + 13;
}
```

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 % 2 == 1 || f(x / 2) < 3)
    return x;
  else
    return x - 2;
}
```

f(2)	f(8)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 11 luglio 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 un elenco di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:

  - `public Integer interoMax()`
    `public Integer interoMin()`
    Restituiscono il riferimento a un oggetto di tipo `Integer` che rappresenta, rispettivamente, il più grande e il più piccolo valore di tipo `Integer` presente nell'elenco. Se l'elenco non contiene alcun oggetto di tipo `Integer` entrambi i metodi restituiscono il riferimento `null`.
  - `public double mediaInteriMinMax()`
    Restituisce un valore *di tipo* `double` uguale alla media tra il più grande e il più piccolo valore di tipo `Integer` presenti nell'elenco. Ad esempio, se il valore più grande è `10` e il più piccolo è `3`, il metodo restituisce `6.5`.

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

2. Riscrivete quanto richiesto per l'esercizio precedente in modo che nel caso l'elenco non contenga alcun oggetto di tipo `Integer` il metodo `mediaInteriMinMax` sollevi una eccezione di tipo `ArithmeticException` (la classe `ArithmeticException` fornisce un costruttore che riceve come argomento una stringa).

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

```
private Number[] numeri
```

che si riferisce ad un array contenente i numeri presenti nell'elenco.
Scrivete l'implementazione del metodo `interoMin`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare gli operatori di confronto come* `<` *e* `>` *a 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`, 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 y;
  private static int w = 2, z = 7;

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

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

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

```
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("balena", 4);
    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 *astratta* `Gamma` che estende `Number` e una classe concreta `Delta` 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:

- ☐ Se il codice sorgente della classe `Delta` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Object` è un supertipo di `Beta`
- ☐ Ogni istanza della classe `Delta` contiene il campo `w`
- ☐ `Gamma` deve fornire l'implementazione del metodo `intValue`
- ☐ Ogni istanza di `Beta` contiene esattamente 3 campi (oltre a quelli ereditati dalla superclasse)
- ☐ `Gamma` è il nome di un tipo riferimento
- ☐ `Delta` deve fornire l'implementazione del metodo `intValue`
- ☐ Ogni oggetto della classe `Beta` contiene esattamente un campo (oltre quelli ereditati dalla superclasse)
- ☐ `Delta` deve fornire l'implementazione dei metodi di `In`
- ☐ `Gamma` deve fornire l'implementazione dei metodi di `In`

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

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

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

- ☐ Il linguaggio Java viene anche chiamato bytecode
- ☐ Le classi astratte sono prive di costruttori
- ☐ Durante l'esecuzione, nello stack si può avere frammentazione
- ☐ La lazy evaluation permette di eseguire in maniera più efficiente le operazioni aritmetiche tra numeri
- ☐ In Java una classe può essere definita estendendo direttamente più classi
- ☐ In Java gli array sono oggetti
- ☐ Il programmatore in Java può definire nuovi tipi riferimento
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Number`, il compilatore segnala un errore
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Le eccezioni di tipo `ArithmeticException` sono controllate
- ☐ L'overriding viene risolto dal compilatore
- ☐ 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 oggetto qualsiasi
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Il tipo `Object` di Java è primitivo
- ☐ Se il codice sorgente di una classe non contiene un costruttore privo di argomenti allora il compilatore lo aggiunge automaticamente
- ☐ Gli oggetti sono memorizzati nello heap
- ☐ I campi statici appartengono alle classi

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

```
int x = 2;
try {
  do
    x = nomi[x - 1].length() / x;
  while (true);
} catch (ArithmeticException e) {
  x = x + 31;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 22;
} catch (NullPointerException e) {
  x = x + 16;
}
```

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) `nomi` contiene `null`. ☐

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

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

(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 % 2 == 1 || f(x / 2) < 3)
    return x;
  else
    return x + 2;
}
```

f(2)	f(8)