Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaInteger()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Integer` presenti nella sequenza.
  - `public int quantiInteger()`
    Restituisce il numero di elementi di tipo `Integer` presenti nella sequenza.
  - `public double mediaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Integer` presenti nella sequenza.

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

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

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

```
private Number[] dati
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Integer` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

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) {
    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 degli 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:

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

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Il tipo `Object` di Java è primitivo
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ 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
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 14;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 4;
} catch (NullPointerException e) {
  x = x + 23;
}
```

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

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

(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 x;
  else
    return (x - 1) * f(x / 2) + 7;
}
```

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public long sommaLong()`
    Restituisce un valore di tipo `long` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Long` presenti nella sequenza.
  - `public int quantiLong()`
    Restituisce il numero di elementi di tipo `Long` presenti nella sequenza.
  - `public double mediaLong()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Long` presenti nella sequenza.

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

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

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

```
private Number[] elementi
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaLong`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Long` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

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) {
    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 degli 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 il campo `x`
- ☐ `Gamma` deve possedere un costruttore privo di argomenti
- ☐ `Number` è un supertipo di `In`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ `Gamma` deve fornire l'implementazione di tutti i metodi di `In`
- ☐ `Gamma` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ È possibile definire una sottoclasse astratta di `Gamma`
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ Il tipo `Object` di Java è primitivo
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ 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 = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 6;
} 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) `nomi` contiene `null`. ☐

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

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

(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 x;
  else
    return (x - 1) * f(x / 2) + 5;
}
```

f(2)	f(5)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaInteger()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Integer` presenti nella sequenza.
  - `public int quantiInteger()`
    Restituisce il numero di elementi di tipo `Integer` presenti nella sequenza.
  - `public double mediaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Integer` presenti nella sequenza.

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

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

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

```
private Number[] valori
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Integer` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

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) {
    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 degli 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:

- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` 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 `In`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ `Gamma` deve fornire l'implementazione di tutti i metodi di `In`
- ☐ Ogni istanza di `Beta` contiene il campo `x`
- ☐ `Gamma` deve possedere un costruttore privo di argomenti
- ☐ È possibile definire una sottoclasse astratta di `Gamma`

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ 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
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ Il tipo `Object` di Java è primitivo
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 15;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 7;
} 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"`, `"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 <= 1)
    return x;
  else
    return (x - 1) * f(x / 2) + 6;
}
```

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaLong()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Long` presenti nella sequenza.
  - `public int quantiLong()`
    Restituisce il numero di elementi di tipo `Long` presenti nella sequenza.
  - `public double mediaLong()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Long` presenti nella sequenza.

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

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

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

```
private Number[] info
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaLong`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Long` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

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) {
    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 degli 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:

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

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ 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
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ 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
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 16;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 18;
} catch (NullPointerException e) {
  x = x + 9;
}
```

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"`, `"ape"`, `""`. ☐

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

(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 x;
  else
    return (x - 1) * f(x / 2) + 4;
}
```

f(2)	f(5)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaInteger()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Integer` presenti nella sequenza.
  - `public int quantiInteger()`
    Restituisce il numero di elementi di tipo `Integer` presenti nella sequenza.
  - `public double mediaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Integer` presenti nella sequenza.

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

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

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

```
private Number[] dati
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Integer` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 1;

  public Beta(int s, int t) {
    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, 10);
    System.out.println(a.intValue());       //2
    a = new Alfa(9);
    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 `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 il campo `x`
- ☐ `Gamma` deve possedere un costruttore privo di argomenti
- ☐ `Number` è un supertipo di `In`
- ☐ È possibile definire una sottoclasse astratta di `Gamma`
- ☐ Se il codice sorgente della classe `Gamma` non contiene un costruttore allora il compilatore segnala un errore
- ☐ `Gamma` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ `Gamma` deve fornire l'implementazione di tutti i metodi di `In`

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ In una stessa classe è possibile definire più metodi con la stesso nome, ma segnatura differente
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ 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*
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Il tipo `Object` di Java è primitivo
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 22;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 17;
} catch (NullPointerException e) {
  x = x + 13;
}
```

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) `nomi` contiene `null`. ☐

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

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

(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 x;
  else
    return (x - 1) * f(x / 2) + 2;
}
```

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public long sommaLong()`
    Restituisce un valore di tipo `long` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Long` presenti nella sequenza.
  - `public int quantiLong()`
    Restituisce il numero di elementi di tipo `Long` presenti nella sequenza.
  - `public double mediaLong()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Long` presenti nella sequenza.

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

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

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

```
private Number[] elementi
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaLong`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Long` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 6;

  public Beta(int s, int t) {
    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(4, 18);
    System.out.println(a.intValue());       //2
    a = new Alfa(12);
    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 `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:

- ☐ `Gamma` deve fornire l'implementazione di tutti i metodi di `In`
- ☐ `Gamma` deve fornire l'implementazione dei metodi astratti di `Number`
- ☐ È possibile definire una sottoclasse astratta di `Gamma`
- ☐ Ogni istanza di `Beta` contiene il campo `x`
- ☐ `Gamma` deve possedere un costruttore privo di argomenti
- ☐ `Number` è un supertipo di `In`
- ☐ `Gamma` deve ridefinire almeno un metodo di `Alfa`
- ☐ Ogni istanza di `Beta` contiene il campo `z`
- ☐ 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

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il tipo `Object` di Java è primitivo
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ 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
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 64;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 25;
} catch (NullPointerException e) {
  x = x + 10;
}
```

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"`, `"ape"`, `""`. ☐

(b) `nomi` contiene `null`. ☐

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

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

f(2)	f(5)

Cognome.....

Nome.....

Matricola.....

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaInteger()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Integer` presenti nella sequenza.
  - `public int quantiInteger()`
    Restituisce il numero di elementi di tipo `Integer` presenti nella sequenza.
  - `public double mediaInteger()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Integer` presenti nella sequenza.

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

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

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

```
private Number[] valori
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaInteger`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come* `+` *a riferimenti di tipo* `Integer` *(non* `Number`*!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 7;

  public Beta(int s, int t) {
    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(9, 6);
    System.out.println(a.intValue());       //2
    a = new Alfa(3);
    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 `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:

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

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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

- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Se una classe contiene più costruttori, almeno uno di questi deve richiamare un costruttore della superclasse
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Se la variabile `a` del metodo `main` di `Prova` viene dichiarata di tipo `Beta`, il compilatore segnala un errore
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ 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
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ Il tipo `Object` di Java è primitivo
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 33;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 44;
} catch (NullPointerException e) {
  x = x + 55;
}
```

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"`, `"ape"`, `""`. ☐

(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 `""`, `"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 <= 1)
    return x;
  else
    return (x - 1) * f(x / 2) + 8;
}
```

f(2)	f(5)

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Programmazione

Prova scritta del 24 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 `SequenzaNumeri`: ogni oggetto della classe rappresenta una sequenza di oggetti `Number`. Tra i metodi forniti dalla classe vi sono:
  - `public int sommaLong()`
    Restituisce un valore di tipo `int` uguale alla somma dei valori rappresentati dagli oggetti di tipo `Long` presenti nella sequenza.
  - `public int quantiLong()`
    Restituisce il numero di elementi di tipo `Long` presenti nella sequenza.
  - `public double mediaLong()`
    Restituisce un valore *di tipo* `double` uguale alla media dei valori di tipo `Long` presenti nella sequenza.

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

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

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

```
private Number[] info
```

che si riferisce ad un array contenente i numeri presenti nella sequenza.
Scrivete l'implementazione del metodo `sommaLong`. *Ricordate che grazie al meccanismo di unboxing è possibile applicare operatori aritmetici come `+` a riferimenti di tipo `Long` (non `Number`!).*

Negli esercizi seguenti supponete di disporre anche di una classe concreta di nome `Alfa`, sottoclasse di `Number`. `Alfa` possiede un costruttore che riceve come argomento un valore di tipo `int` e un costruttore privo di argomenti. 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`. La chiamata del costruttore privo di argomenti produce un oggetto uguale a quello ottenuto mediante la chiamata `new Alfa(13)`.

4. Considerate le seguenti classi:

```
public class Beta extends Alfa {
  private int x, y;
  private static int z = 8;

  public Beta(int s, int t) {
    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(4, 2);
    System.out.println(a.intValue());       //2
    a = new Alfa(15);
    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 `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:

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

b. Considerate le seguenti dichiarazioni di variabile: `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

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

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
- ☐ Durante l'esecuzione lo stack può contenere, nello stesso momento, record di attivazione di costruttori differenti
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in compilazione*
- ☐ La variabile `a` del metodo `main` della classe `Prova` viene memorizzata nello heap
- ☐ In Java ogni classe (incluse le classi astratte) possiede un costruttore
- ☐ Il costrutto di *selezione* permette di ripetere l'esecuzione di un blocco di istruzioni in base a una condizione
- ☐ Gli oggetti di una classe dichiarata `final` non possono essere modificati
- ☐ Durante l'esecuzione lo stack può contenere più record di attivazione di uno stesso costruttore
- ☐ Il costrutto `try-catch` permette di *intercettare* eccezioni
- ☐ Il costruttore di `Beta` richiama un costruttore della superclasse `Alfa`
- ☐ Il costrutto `try-catch` permette di *sollevare* eccezioni
- ☐ I riferimenti agli oggetti sono sempre memorizzati nello heap
- ☐ 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
- ☐ In Java ogni classe possiede un costruttore privo di argomenti
- ☐ Gli oggetti sono sempre memorizzati nello stack
- ☐ 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
- ☐ I campi statici, come `z` della classe `Beta`, sono memorizzati nello heap
- ☐ Sostituendo `super` con `this` nel codice del metodo `intValue` della classe `Beta` si avrà un errore *in esecuzione*

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

```
int x = 2;
try {
  x = nomi[nomi[x].length()].length() / nomi[x].length();
} catch (ArithmeticException e) {
  x = x + 11;
} catch (ArrayIndexOutOfBoundsException e) {
  x = x + 51;
} catch (NullPointerException e) {
  x = x + 43;
}
```

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"`, `"ape"`, `""`. ☐

(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 `""`, `"cane"`, `"ape"`, `"formica"`. ☐

(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 x;
  else
    return (x - 1) * f(x / 2) + 9;
}
```

f(2)	f(5)