

Settembre 2025

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(long l) {return "P.m(long value)";}  
    String m(Number n) {return "P.m(number)";}  
}  
public class H extends P {  
    String m(double d) {return "H.m(double value)";}  
    String m(Number n) {return "H.m(number)";}  
}  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42L)`
- (b) `p2.m(42L)`
- (c) `h.m(42L)`
- (d) `p.m(42D)`
- (e) `p2.m(42D)`
- (f) `h.m(42D)`

Luglio 2025

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

i. Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(int)` dichiarato in `P` è applicabile per sottotipo, poiché `int` $\not\leq$ `Object` e `int` \leq `int`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(int)` in `P` e viene stampata la stringa "`P.m(int value)`".

ii. L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(int)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e stampata la stringa "`P.m(int value)`".

iii. Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(int)` che è applicabile per le stesse ragioni dei punti precedenti. Dei metodi dichiarati in `H` solo `m(float)` è applicabile perché `int` $\not\leq$ `Object` e `int` \leq `float`. Dei due, `m(int)` è il più specifico poiché `int` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo ereditato da `P` e stampata la stringa "`P.m(int value)`".

iv. Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo, poiché `long` $\not\leq$ `Object` e `long` \leq `int`;
- secondo tentativo (boxing/unboxing e sottotipo): l'argomento può essere convertito nel tipo `Long` per boxing. Dopo tale conversione solo il metodo `m(Object)` è applicabile per sottotipo poiché `Long` \leq `Object` e `long` \leq `int`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Object)` in `P` e viene stampata la stringa "`P.m(object)`".

v. L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Object)` è ridefinito in `H`, viene stampata la stringa "`H.m(object)`".

vi. Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo `m(int)` ereditato da `P` non è applicabile per le stesse ragioni dei punti precedenti. Dei metodi dichiarati in `H`, solo `m(float)` è applicabile, poiché `long` $\not\leq$ `Object` e `long` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

a) Il literal `42f` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(float)` dichiarato in `P` è applicabile per sottotipo, poiché `float` $\not\leq$ `Object` e `float` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene stampata la stringa "`H.m(float value)`".

c) Il literal `42f` ha tipo statico `float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Object)` che non è applicabile per le stesse ragioni dei punti precedenti. Entrambi i metodi dichiarati in `H` sono applicabili poiché `float` \leq `float` e `float` \leq `double`. Dei due, `m(float)` è il più specifico poiché `float` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

d) Il literal `42d` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo, poiché `double` $\not\leq$ `Object` e `double` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Object)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e, quindi, viene stampata la stringa "`P.m(object)`".

e) Il literal `42d` ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo `m(double)` ereditato da `P` non è applicabile per le stesse ragioni dei punti precedenti. Dei metodi dichiarati in `H`, solo `m(double)` è applicabile, poiché `double` \leq `float` e `double` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(double)` di `H`; viene stampata la stringa "`H.m(double value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

a) Il literal `42f` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(float)` dichiarato in `P` è applicabile per sottotipo, poiché `float` $\not\leq$ `Object` e `float` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene stampata la stringa "`H.m(float value)`".

c) Il literal `42f` ha tipo statico `float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Object)` che non è applicabile per le stesse ragioni dei punti precedenti. Entrambi i metodi dichiarati in `H` sono applicabili poiché `float` \leq `float` e `float` \leq `double`. Dei due, `m(float)` è il più specifico poiché `float` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

d) Il literal `42d` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo, poiché `double` $\not\leq$ `Object` e `double` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Object)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e, quindi, viene stampata la stringa "`P.m(object)`".

e) Il literal `42d` ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo `m(double)` ereditato da `P` non è applicabile per le stesse ragioni dei punti precedenti. Dei metodi dichiarati in `H`, solo `m(double)` è applicabile, poiché `double` \leq `float` e `double` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(double)` di `H`; viene stampata la stringa "`H.m(double value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

a) Il literal `42f` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(float)` dichiarato in `P` è applicabile per sottotipo, poiché `float` $\not\leq$ `Object` e `float` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene stampata la stringa "`H.m(float value)`".

c) Il literal `42f` ha tipo statico `float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Object)` che non è applicabile per le stesse ragioni dei punti precedenti. Entrambi i metodi dichiarati in `H` sono applicabili poiché `float` \leq `float` e `float` \leq `double`. Dei due, `m(float)` è il più specifico poiché `float` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

a) Il literal `42f` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(float)` dichiarato in `P` è applicabile per sottotipo, poiché `float` $\not\leq$ `Object` e `float` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene stampata la stringa "`H.m(float value)`".

c) Il literal `42f` ha tipo statico `float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Object)` che non è applicabile per le stesse ragioni dei punti precedenti. Entrambi i metodi dichiarati in `H` sono applicabili poiché `float` \leq `float` e `float` \leq `double`. Dei due, `m(float)` è il più specifico poiché `float` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

a) Il literal `42f` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(float)` dichiarato in `P` è applicabile per sottotipo, poiché `float` $\not\leq$ `Object` e `float` \leq `float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene stampata la stringa "`H.m(float value)`".

c) Il literal `42f` ha tipo statico `float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Object)` che non è applicabile per le stesse ragioni dei punti precedenti. Entrambi i metodi dichiarati in `H` sono applicabili poiché `float` \leq `float` e `float` \leq `double`. Dei due, `m(float)` è il più specifico poiché `float` \leq `double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

febbraio 2025

Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(Long l) {return "P.m(long object)";}  
    String m(Double d) {return "P.m(double object)";}  
}  
  
public class H extends P {  
    String m(long l) {return "H.m(long value)";}  
    String m(Double d) {return "H.m(double object)";}  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42L)`
- (b) `p2.m(42L)`
- (c) `h.m(42L)`
- (d) `p.m(42.0)`
- (e) `p2.m(42.0)`
- (f) `h.m(42.0)`

gennaio 2025 (+ parziale)

Soluzione: visto che si assume che tutte le classi sono dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): dei due metodi dichiarati in `P` è applicabile per sottotipo solo `m(float)` poiché `int <= float & int <= Integer`. Tale metodo è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` dichiarato in `P` e viene stampata la stringa `"P.m(float value)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene eseguito il metodo `m(float)` dichiarato in `H` e stampata la stringa `"H.m(float value)"`.

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo `m(float)` dichiarato in `H` è l'unico applicabile per sottotipo poiché `int <= float & int <= Integer`, mentre il metodo `m(Integer)` di `P` ereditato da `H` e il metodo `m(Long)` dichiarato in `H` non sono applicabili per sottotipo poiché `int <= Integer, int <= Long`.

Essendo l'unico metodo applicabile, `m(float)` è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` dichiarato in `H` e viene stampata la stringa `"H.m(float value)"`.

(d) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): dei due metodi dichiarati in `P` è applicabile per sottotipo solo `m(float)` poiché `long <= float & long <= Integer`. Tale metodo è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` dichiarato in `P` e viene stampata la stringa `"P.m(float value)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene eseguito il metodo `m(float)` dichiarato in `H` e stampata la stringa `"H.m(float value)"`.

(f) L'argomento `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo `m(float)` dichiarato in `H` è l'unico applicabile per sottotipo poiché `long <= float & long <= Integer`, mentre il metodo `m(Integer)` di `P` ereditato da `H` e il metodo `m(Long)` dichiarato in `H` non sono applicabili per sottotipo poiché `long <= Integer, long <= Long`.

Essendo l'unico metodo applicabile, `m(float)` è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` dichiarato in `H` e viene stampata la stringa `"H.m(float value)"`.

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(int i) {return "P.m(int value)";}  
    String m(Float f) {return "P.m(float object)";}  
}  
  
public class H extends P {  
    String m(long l) {return "H.m(long value)";}  
    String m(Float f) {return "H.m(float object)";}  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42F)`
- (e) `p2.m(42F)`
- (f) `h.m(42F)`

Soluzione: visto che si assume che tutte le classi sono dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessuno dei due metodi dichiarati in `P` è applicabile per sottotipo poiché `long <= Long & long <= Double`.

- secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42L` può essere convertito per boxing a `Long`, quindi solo il metodo `m(Long)` è applicabile per sottotipo, poiché `Long <= Long & Long <= Double`. Il metodo è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Long)` in `P` e viene stampata la stringa `"P.m(long object)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Long)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e viene stampata la stringa `"P.m(Long object)"`.

(c) Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo `m(long)` dichiarato in `H` è l'unico applicabile per sottotipo poiché ovviamente `long <= long`, mentre il metodo `m(Long)` di `P` ereditato da `H` e il metodo `m(Double)` dichiarato in `H` non sono applicabili per sottotipo poiché `long <= Long, long <= Double`.

Essendo l'unico metodo applicabile, `m(long)` è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(long)` di `H` e viene stampata la stringa `"H.m(long value)"`.

(d) Il literal `42.0` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo dato che `double <= Long & double <= Double`.

- secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42.0` può essere convertito per boxing a `Double`, quindi solo il metodo `m(Double)` è applicabile per sottotipo, poiché `Double <= Long, Double <= Double`.

Il metodo è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Double)` in `P` e viene stampata la stringa `"P.m(double object)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Double)` è ridefinito in `H`, viene eseguita la versione di `H` e viene stampata la stringa `"H.m(double object)"`.

(f) L'argomento `42.0` ha tipo statico `Double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): nessuno dei metodi dichiarati in `H`, né quello ereditato da `P`, è applicabile per sottotipo dato che `Double <= Long, double <= long & double <= Double`.

- secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42.0` può essere convertito per boxing a `Double`.

Il metodo `m(Long)` di `P` ereditato da `H` e il metodo `m(Long)` definito in `H` non sono applicabili per sottotipo poiché `Double <= Long & Double <= long` mentre il metodo `m(Double)` dichiarato in `H` è ovviamente applicabile per sottotipo poiché `Double <= Double` ed è anche il metodo più specifico.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Double)` di `H` e viene stampata la stringa `"H.m(double object)"`.

. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(float f) {return "P.m(float value)";}  
    String m(Integer i) {return "P.m(int object)";}  
}  
  
public class H extends P {  
    String m(float f) {return "H.m(float value)";}  
    String m(Long l) {return "H.m(long object)";}  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p.m(42)`

(b) `p2.m(42)`

(c) `h.m(42)`

(d) `p.m(42L)`

(e) `p2.m(42L)`

(f) `h.m(42L)`

Soluzione: visto che si assume che tutte le classi sono dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): dei due metodi dichiarati in `P`, solo `m(int)` è applicabile per sottotipo poiché `int <= int & int <= Float`. Ovviamente tale metodo è anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(int)` in `P` e viene stampata la stringa `"P.m(int value)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(int)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e viene stampata la stringa `"P.m(int value)"`.

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo `m(int)` di `P` ereditato da `H` e il metodo `m(long)` dichiarato in `H` sono gli unici applicabili per sottotipo poiché `int <= int, int <= long & int <= Float`.

Il più specifico è dato che `int <= long`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` ereditato da `P` e viene stampata la stringa `"P.m(int value)"`.

(d) Il literal `42F` ha tipo statico `float` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo dato che `float <= int & float <= Float`.

- secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42F` può essere convertito per boxing a `Float`, quindi solo il metodo `m(Float)` della classe `P` è applicabile per sottotipo, poiché `Float <= float & float <= Float`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Float)` in `P` e viene stampata la stringa `"P.m(float object)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene eseguita la versione di `H` e viene stampata la stringa `"H.m(float object)"`.

(f) L'argomento `42F` ha tipo statico `Float` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo `m(int)` di `P` ereditato da `H` e entrambi i metodi dichiarati in `H` non sono applicabili per sottotipo dato che `float <= int, float <= long & float <= Float`.

- secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42F` può essere convertito per boxing a `Float`.

Il metodo `m(int)` di `P` ereditato da `H` e il metodo `m(long)` definito in `H` non sono applicabili per sottotipo poiché `Float <= int & float <= long` mentre il metodo `m(Float)` dichiarato in `H` è ovviamente applicabile per sottotipo poiché `Float <= float`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Float)` di `H` e viene stampata la stringa `"H.m(float object)"`.

Settembre 2024

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Double d) { return "P.m(double object)"; }
    String m(Long l) { return "P.m(long value)"; }
}
public class H extends P {
    String m(Double d) { return "H.m(double object)"; }
    String m(Float f) { return "H.m(float value)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42.)`
- (e) `p2.m(42.)`
- (f) `h.m(42.)`

Soluzione: visto che si assume che tutte le classi sono dichiarate nello stesso package, tutti i metodi sono accessibili.

- (a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): dei due metodi dichiarati in `P`, solo `m(long)` è applicabile per sottotipo poiché `int` \leq `long` e `int` $\not\leq$ `Double`. Ovviamente tale metodo è anche il più specifico.A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa "`P.m(long value)`".
- (b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
 - A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(long)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e viene stampata la stringa "`P.m(long value)`".
- (c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): il metodo `m(long)` di `P` ereditato da `H` e il metodo `m(float)` dichiarato in `H` sono gli unici applicabili per sottotipo poiché `int` \leq `long`, `int` \leq `float` e `int` $\not\leq$ `Double`. Il più specifico è `m(long)` dato che `long` \leq `float`.A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(long)` ereditato da `P` e viene stampata la stringa "`P.m(long value)`".
- (d) Il literal `42.` ha tipo statico `double` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo dato che `double` $\not\leq$ `long` e `double` $\not\leq$ `Double`.
 - secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42.` può essere convertito per boxing a `Double`, quindi solo il metodo `m(Double)` della classe `P` è applicabile per sottotipo, poiché `Double` \leq `Double` e `Double` $\not\leq$ `long`. Il metodo è ovviamente anche il più specifico.A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Double)` in `P` e viene stampata la stringa "`P.m(Double object)`".
- (e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
 - A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Double)` è ridefinito in `H`, viene eseguita la versione di `H` e viene stampata la stringa "`H.m(Double object)`".
- (f) L'argomento `42.` ha tipo statico `double` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): il metodo `m(long)` di `P` ereditato da `H` e entrambi i metodi dichiarati in `H` non sono applicabili per sottotipo dato che `double` $\not\leq$ `long`, `double` $\not\leq$ `Double` e `double` $\not\leq$ `float`.
 - secondo tentativo (boxing/unboxing e sottotipo): l'argomento `42.` può essere convertito per boxing a `Double`. Il metodo `m(long)` di `P` ereditato da `H` e il metodo `m(float)` definito in `H` non sono applicabili per sottotipo poiché `Double` $\not\leq$ `long` e `Double` $\not\leq$ `float` mentre il metodo `m(Double)` dichiarato in `H` è ovviamente applicabile per sottotipo poiché `Double` \leq `Double` ed è anche il metodo più specifico.A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Double)` di `H` e viene stampata la stringa "`H.m(Double object)`".

Luglio 2024

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Float f) { return "P.m(float value)"; }
    String m(Int i) { return "P.m(int value)"; }
}
public class H extends P {
    String m(Float f) { return "H.m(float value)"; }
    String m(double d) { return "H.m(double value)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(Float.valueOf(42F))`
- (e) `p2.m(Float.valueOf(42F))`
- (f) `h.m(Float.valueOf(42F))`

Soluzione: visto che si assume che tutte le classi sono dichiarate nello stesso package, tutti i metodi sono accessibili.

- (a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): entrambi i metodi dichiarati in `P` sono applicabili per sottotipo, poiché `int` \leq `float` e `int` \leq `int`. Il più specifico è `m(int)` dato che `int` \leq `float`.A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(int)` in `P` e viene stampata la stringa "`P.m(int value)`".
- (b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
 - A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(int)` non è ridefinito in `H`, viene eseguita la versione di `H` e viene stampata la stringa "`H.m(int value)`".
- (c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): il metodo `m(int)` di `P` ereditato da `H` e entrambi i metodi dichiarati in `H` sono applicabili per sottotipo poiché `int` \leq `int`, `int` \leq `float` e `int` \leq `double`. Il più specifico è `m(int)` dato che `int` \leq `float` e `int` \leq `double`.A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` ereditato da `P` e viene stampata la stringa "`P.m(int value)`".
- (d) Il literal `42F` ha tipo statico `float` e il tipo statico di `p` è `P`.La classe `Float` contiene solo le due versioni `Float.valueOf(float)` e `Float.valueOf(String)` del metodo di classe `valueOf`. Solo la prima è applicabile per sottotipo dato che `float` \leq `float` e `float` $\not\leq$ `String`, quindi l'argomento `Float.valueOf(42F)` ha tipo `Float`.
 - primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo dato che `Float` $\not\leq$ `float` e `Float` $\not\leq$ `int`.
 - secondo tentativo (boxing/unboxing e sottotipo): l'argomento `Float.valueOf(42F)` può essere convertito per unboxing a `float`, quindi solo il metodo `m(float)` della classe `P` è applicabile per sottotipo, poiché `float` \leq `float` e `float` $\not\leq$ `int`.A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".
- (e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
 - A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(float)` è ridefinito in `H`, viene eseguito il metodo `m(float)` di `H` e viene stampata la stringa "`H.m(float value)`".
- (f) L'argomento `Float.valueOf(42F)` ha tipo statico `Float` per gli stessi motivi del punto (d) e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): il metodo `m(int)` di `P` ereditato da `H` e entrambi i metodi dichiarati in `H` non sono applicabili per sottotipo dato che `Float` $\not\leq$ `int`, `Float` $\not\leq$ `float` e `Float` $\not\leq `double`.$
 - secondo tentativo (boxing/unboxing e sottotipo): l'argomento `Float.valueOf(42F)` può essere convertito per unboxing a `float`. Il metodo `m(int)` di `P` ereditato da `H` non è applicabile per sottotipo poiché `Float` $\not\leq$ `int` mentre entrambi i metodi dichiarati in `H` sono applicabili per sottotipo poiché `float` \leq `float` e `float` \leq `double`. Il metodo più specifico è `m(float)` dato che `float` \leq `double`.A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H` e viene stampata la stringa "`H.m(float value)`".

giugno 2024

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(Double d) { return "P.m(double object)"; }  
    String m(long l) { return "P.m(long value)"; }  
}  
  
public class H extends P {  
    String m(Number n) { return "H.m(number object)"; }  
    String m(long l) { return "H.m(long value)"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42L)`
- (e) `p2.m(42L)`
- (f) `h.m(42L)`

febbraio 2024 (+ parziale)

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(Long l) { return "P.m(long object)"; }  
    String m(Integer i) { return "P.m(integer object)"; }  
}  
  
public class H extends P {  
    String m(Long l) { return "H.m(long object)"; }  
    String m(int i) { return "H.m(int value)"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42L)`
- (e) `p2.m(42L)`
- (f) `h.m(42L)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(long)` della classe `P` è accessibile e applicabile per sottotipo, poiché `int ≤ long` e `int ≤ Integer`; quindi è anche il metodo più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa `"P.m(long value)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e, poiché il metodo `m(long)` è ridefinito in `H`, viene eseguito il metodo `m(long)` di `H`, quindi, viene stampata la stringa `"H.m(long value)"`.

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo ereditato da `H`, `m(Integer)`, è accessibile ma non applicabile per sottotipo per il motivo del punto (a). I metodi `m(long)` e `m(int)` definiti in `H` sono entrambi accessibili e applicabili per sottotipo poiché `int ≤ long` e `int ≤ int`. Dato che `int ≤ long` viene scelto il metodo più specifico `m(int)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` di `H`; viene stampata la stringa `"H.m(int value)"`.

(d) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`. Per quanto riguarda il metodo `valueOf`, la classe `Long` ha solo i due metodi `Long.valueOf(long)` e `Long.valueOf(String)` che sono entrambi pubblici e quindi accessibili. Poiché `long ≤ long` e `long ≤ String`, solo `valueOf(long)` è applicabile a `42L` per sottotipo, quindi `Long.valueOf(42L)` ha tipo `Long`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è accessibile e applicabile per sottotipo, poiché `Long ≤ long` e `Long ≤ Integer`;
- secondo tentativo (boxing/unboxing e opzionalmente sottotipo): tramite unboxing l'argomento può essere convertito al tipo `long`; quindi per gli stessi motivi del punto (a) la chiamata viene risolta con il metodo `m(long)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi per lo stesso motivo del punto (a) viene stampata la stringa `"P.m(long value)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` quindi per gli stessi motivi del punto (b) viene stampata la stringa `"H.m(long value)"`.

(f) Per i motivi del punto (d) `Long.valueOf(42L)` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): per lo stesso motivo del punto (a), l'unico metodo `m(Integer)` ereditato da `P` è accessibile ma non è applicabile per sottotipo; anche i due metodi definiti in `H` sono accessibili ma non applicabili per sottotipo poiché `long ≤ long` e `long ≤ int`;
- secondo tentativo (boxing/unboxing e opzionalmente sottotipo): come per il punto (e) tramite unboxing l'argomento può essere convertito al tipo `long`; quindi per gli stessi motivi del punto (a) la chiamata viene risolta con il metodo `m(long)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per gli stessi motivi del punto (b) viene stampata la stringa `"H.m(long value)"`.

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): solo il metodo `m(long)` dichiarato in `P` è applicabile per sottotipo, poiché `int ≤ Double` e `int ≤ long`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa `"P.m(long value)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(long)` è ridefinito in `H`, viene stampata la stringa `"H.m(long value)"`.

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): l'unico metodo di `P` ereditato da `H` è `m(Double)` che non è applicabile per le stesse ragioni dei punti precedenti. L'unico dei metodi dichiarati in `H` applicabile è `m(long)` poiché `int ≤ long` e `int ≤ Number`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(long)` di `H`; viene stampata la stringa `"H.m(long value)"`.

(d) Il literal `42.0` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): nessun metodo della classe `P` è applicabile per sottotipo, poiché `double ≤ Double` e `double ≤ long`;
- secondo tentativo (boxing/unboxing e sottotipo): l'argomento può essere convertito al tipo `Double` per boxing. Dopo tale conversione solo il metodo `m(Double)` è applicabile per sottotipo poiché `Double ≤ Double` e `Double ≤ long`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Double)` in `P` e viene stampata la stringa `"P.m(double object)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Double)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e, quindi, viene stampata la stringa `"P.m(double object)"`.

(f) Il literal `42.0` ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): nessun metodo dichiarato in `H` o ereditato da `P` è applicabile per sottotipo dato che `double ≤ long`, `double ≤ Double` e `double ≤ Number`;
- secondo tentativo (boxing/unboxing e sottotipo): l'argomento può essere convertito al tipo `Double` per boxing. Dopo tale conversione gli unici metodi applicabili per sottotipo sono `m(Double)` e `m(Number)` poiché `Double ≤ long`, `Double ≤ Double` e `Double ≤ Number`. Viene selezionato il metodo più specifico `m(Double)` dato che `Double ≤ Number`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Double)` di `P`; viene stampata la stringa `"P.m(long object)"`.

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): entrambi i metodi accessibili della classe `P` non sono applicabili per sottotipo, poiché `int ≤ Long` e `int ≤ Integer`;
- secondo tentativo (boxing/unboxing e sottotipo): l'argomento può essere convertito al tipo `Integer` per boxing. Solo il metodo `m(Integer)` è applicabile per sottotipo e, quindi, viene scelto, poiché `Integer ≤ Long` e `Integer ≤ Integer`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Integer)` in `P` e viene stampata la stringa `"P.m(integer object)"`.

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Integer)` non è ridefinito in `H`, viene eseguito quello ereditato da `P` e, quindi, viene stampata la stringa `"P.m(integer object)"`.

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(Long)` che è accessibile ma non applicabile per sottotipo per lo stesso motivo dei punti precedenti; dei metodi accessibili definiti in `H` solo `m(int)` è applicabile per sottotipo e, quindi, viene scelto, poiché `int ≤ Long` e `int ≤ int`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` di `H`; viene stampata la stringa `"H.m(int value)"`.

(d) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): entrambi i metodi accessibili della classe `P` non sono applicabili per sottotipo, poiché `long ≤ Long` e `long ≤ Integer`;
- secondo tentativo (boxing/unboxing e sottotipo): l'argomento può essere convertito al tipo `Long` per boxing. Solo il metodo `m(Long)` è applicabile per sottotipo e, quindi, viene scelto, poiché `Long ≤ Long` e `Long ≤ Integer`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Long)` in `P` e viene stampata la stringa `"P.m(long object)"`.

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(Long)` è ridefinito in `H`, viene eseguito quest'ultimo e, quindi, viene stampata la stringa `"H.m(long object)"`.

(f) Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(Integer)` che è accessibile ma non applicabile per sottotipo dato che `long ≤ Integer`; dei due metodi accessibili definiti in `H` solo `m(Long)` è applicabile per sottotipo poiché `Long ≤ Long` e `Long ≤ int`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Long)` di `H`; viene stampata la stringa `"H.m(long object)"`.

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(Long l) { return "P.m(long value)"; }  
    String m(Integer i) { return "P.m(integer object)"; }  
}  
  
public class H extends P {  
    String m(Long l) { return "H.m(long value)"; }  
    String m(int i) { return "H.m(int value)"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(Long.valueOf(42L))`
- (e) `p2.m(Long.valueOf(42L))`
- (f) `h.m(Long.valueOf(42L))`

giugno 2023

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(double d) { return "P.m(double value)"; }
    String m(long l) { return "P.m(long value)"; }
}

public class H extends P {
    String m(double d) { return "H.m(double value)"; }
    String m(int i) { return "H.m(int value)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42.0F)`
- (e) `p2.m(42.0F)`
- (f) `h.m(42.0F)`

febbraio 2023 (+ parziale)

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(double d) { return "P.m(double value)"; }
    String m(int i) { return "P.m(int value)"; }
}

public class H extends P {
    String m(float f) { return "H.m(float value)"; }
    String m(int i) { return "H.m(int value)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42.0F)`
- (e) `p2.m(42.0F)`
- (f) `h.m(42.0F)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

- (a) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): poiché `long` ≤ `double` e `long` ≠ `Double`, l'unico metodo di `P` applicabile per sottotipo è `m(double)`.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double)` in `P` e viene stampata la stringa "`P.m(double value)`".
- (b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(double)` è ridefinito in `H`, viene eseguito il metodo della classe `H` e, quindi, viene stampata la stringa "`H.m(double value)`".
- (c) Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): dei due metodi ereditati da `P`, l'unico applicabile per sottotipo è `m(double)`, come ai punti precedenti; i metodi definiti in `H` sono entrambi applicabili per sottotipo, poiché `long` ≤ `float` e `long` ≤ `double`, quindi viene scelto `m(float)` poiché più specifico di `m(double)`, in quanto `float` ≤ `double`.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H` e viene stampata la stringa "`H.m(float value)`".
- (d) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): poiché `float` ≤ `double` e `float` ≠ `Double`, l'unico metodo di `P` applicabile per sottotipo è `m(double)`.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double)` in `P` e viene stampata la stringa "`P.m(double value)`".
- (e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(double)` è ridefinito in `H`, viene eseguito il metodo della classe `H` e, quindi, viene stampata la stringa "`H.m(double value)`".
- (f) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): dei due metodi ereditati da `P`, l'unico applicabile per sottotipo è `m(double)`, come ai punti precedenti; i metodi definiti in `H` sono entrambi applicabili per sottotipo, poiché `float` ≤ `float` e `float` ≤ `double`, quindi viene scelto `m(float)` poiché più specifico di `m(double)`, in quanto `float` ≤ `double`.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H` e viene stampata la stringa "`H.m(float value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

- (a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): entrambi i metodi della classe `P` sono accessibili e applicabili per sottotipo, poiché `int` ≤ `long` ≤ `double`; dato che `long` ≤ `double` viene scelto il metodo più specifico `m(long)`.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa "`P.m(long value)`".
- (b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(long)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e, quindi, viene stampata la stringa "`P.m(long value)`".
- (c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(long)` che è accessibile e applicabile per sottotipo dato che `int` ≤ `long`; anche entrambi i metodi definiti in `H` sono accessibili e applicabili per sottotipo poiché `int` ≤ `int` ≤ `int`. Dato che `int` ≤ `long` ≤ `double` viene scelto il metodo più specifico `m(int)`.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` di `H`; viene stampata la stringa "`H.m(int value)`".
- (d) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): solo il metodo `m(double)` della classe `P` è accessibile e applicabile per sottotipo dato che `float` ≤ `double`; quindi `m(double)` è anche il metodo più specifico.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double)` in `P` e viene stampata la stringa "`P.m(double value)`".
- (e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(double)` è ridefinito in `H`, viene eseguito il metodo ereditato da `H` e, quindi, viene stampata la stringa "`H.m(double value)`".
- (f) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(long)` che è accessibile ma non è applicabile per sottotipo dato che `float` ≤ `long`; dei due metodi definiti in `H` solo `m(double)` è accessibile e applicabile per sottotipo poiché `float` ≤ `double` e `float` ≤ `int`; quindi è anche il metodo più specifico.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(double)` di `H`; viene stampata la stringa "`H.m(double value)`".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

- (a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): entrambi i metodi della classe `P` sono accessibili e applicabili per sottotipo, poiché `int` ≤ `double` e `int` ≤ `int`; dato che `int` ≤ `double` viene scelto il metodo più specifico `m(int)`.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(int)` in `P` e viene stampata la stringa "`P.m(int value)`".
- (b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo `m(int)` è ridefinito in `H`, viene eseguito quest'ultimo e, quindi, viene stampata la stringa "`H.m(int value)`".
- (c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(double)` che è accessibile e applicabile per sottotipo dato che `int` ≤ `double`; anche entrambi i metodi definiti in `H` sono accessibili e applicabili per sottotipo poiché `int` ≤ `float` e `int` ≤ `int`. Dato che `int` ≤ `double` e `int` ≤ `float` viene scelto il metodo più specifico `m(int)`.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` di `H`; viene stampata la stringa "`H.m(int value)`".
- (d) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `p` è `P`.
 - primo tentativo (solo sottotipo): solo il metodo `m(double)` della classe `P` è accessibile e applicabile per sottotipo, poiché `float` ≤ `double` e `float` ≤ `int`; quindi `m(double)` è anche il metodo più specifico.
A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double)` in `P` e viene stampata la stringa "`P.m(double value)`".
- (e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.
A runtime, il tipo dinamico dell'oggetto in `p2` è `H` ma poiché il metodo `m(double)` non è ridefinito in `H`, viene eseguito il metodo ereditato da `P` e, quindi, viene stampata la stringa "`P.m(double value)`".
- (f) Il literal `42.0F` ha tipo statico `float` e il tipo statico di `h` è `H`.
 - primo tentativo (solo sottotipo): `H` eredita da `P` solamente il metodo `m(double)` che è accessibile e applicabile per sottotipo dato che `float` ≤ `double`; dei due metodi definiti in `H` solo `m(float)` è accessibile e applicabile per sottotipo e, quindi, viene selezionato.
A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; viene stampata la stringa "`H.m(float value)`".

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(double d) { return "P.m(double value)"; }
    String m(Double d) { return "P.m(double object)"; }
}

public class H extends P {
    String m(float f) { return "H.m(float value)"; }
    String m(double d) { return "H.m(double value)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42L)`
- (b) `p2.m(42L)`
- (c) `h.m(42L)`
- (d) `p.m(42.0F)`
- (e) `p2.m(42.0F)`
- (f) `h.m(42.0F)`

gennaio 2023

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(float f) { return "P.m(float value)"; }
    String m(Float f) { return "P.m(Float Object)"; }
}
public class H extends P {
    String m(float f) { return "H.m(Float Object)"; }
    String m(Number n) { return "H.m(Number)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42L)`
- (b) `p2.m(42L)`
- (c) `h.m(42L)`
- (d) `p.m(Float.valueOf(42.0F))`
- (e) `p2.m(Float.valueOf(42.0F))`
- (f) `h.m(Float.valueOf(42.0F))`

settembre 2022

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `int` \leq `long` \leq `Long`, l'unico metodo di `P` applicabile per sottotipo è `m(long)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa "`P.m(long value)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(long)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(long value)`".

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): dei due metodi ereditati da `P`, l'unico applicabile per sottotipo è `m(long)`, come ai punti precedenti; i metodi definiti in `H` non sono applicabili per sottotipo, poiché `int` \leq `Long` \leq `Number`, quindi viene scelto `m(long)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(long)` ereditato da `P` e viene stampata la stringa "`P.m(long value)`".

(d) Il literal `42.0` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `double` \leq `long` \leq `Long`, nessun metodo di `P` è applicabile per sottotipo.
- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing `double` potrebbe essere convertito a `Double`, ma poiché `Double` \leq `long` \leq `Long`, nessun metodo è applicabile.
- terzo tentativo (numero variabile di parametri): non essendo dichiarati metodi con numero variabile di parametri, nessun metodo è applicabile, quindi il codice non è staticamente corretto.

(e) Visto che i tipi statici sono gli stessi del punto precedente, il codice non è staticamente corretto.

(f) Il literal `42.0` ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): i metodi ereditati da `P` non sono applicabili per sottotipo analogamente ai punti precedenti; inoltre, nessuno dei due metodi definiti in `H` è applicabile per sottotipo poiché `double` \leq `long` \leq `Long`.
- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing `double` può essere convertito a `Double`; in questo modo l'unico metodo applicabile è `m(Number)`, poiché `Double` \leq `long`, `Double` \leq `Long` e `Double` \leq `Number`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Number)` di `H` e viene stampata la stringa "`H.m(Number)`".

luglio 2022

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Number n) { return "P.m(Number)"; }
    String m(Long l) { return "P.m(Long Object)"; }
}
public class H extends P {
    String m(Long l) { return "H.m(Long Object)"; }
    String m(Long l) { return "H.m(Long value)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42L)`
- (e) `p2.m(42L)`
- (f) `h.m(42L)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `long` \leq `float` \leq `Float`, l'unico metodo di `P` applicabile per sottotipo è `m(float)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float value)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(float)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(float value)`".

(c) Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): dei due metodi ereditati da `P`, l'unico applicabile per sottotipo è `m(float)`, come ai punti precedenti; i metodi definiti in `H` non sono applicabili per sottotipo, poiché `long` \leq `Float` \leq `long` \leq `Number`, quindi viene scelto `m(float)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` ereditato da `P` e viene stampata la stringa "`P.m(float value)`".

(d) Il literal `42.0F` ha tipo statico `float`, la classe `Float` ha due soli metodi `valueOf(float)`: `valueOf(float)` e `valueOf(String)`; solo il primo è applicabile per sottotipo dato che `float` \leq `float` \leq `String`. Poiché `valueOf(float)` ha tipo di ritorno `Float`, il tipo statico di `Float.valueOf(42.0F)` è `Float`, mentre il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `Float` \leq `float` \leq `Float` \leq `Float`, l'unico metodo di `P` applicabile per sottotipo è `m(float)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float Object)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, quindi viene eseguito il metodo `m(float)` ridefinito in `H` e viene stampata la stringa "`H.m(float Object)`".

(f) L'espressione `Float.valueOf(42.0F)` ha tipo statico `Float` come nei due punti precedenti e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): il metodo ereditato da `P` non è applicabile, come ai punti precedenti; i due metodi definiti in `H` sono entrambi applicabili dato che `Float` \leq `Float` \leq `Number` e il più specifico è `m(float)` poiché `Float` \leq `Number`, quindi viene scelto `m(float)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(Float)` di `H` e viene stampata la stringa "`H.m(Float Object)`".

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(long l) { return "P.m(long value)"; }
    String m(Long l) { return "P.m(Long Object)"; }
}
public class H extends P {
    String m(Long l) { return "H.m(Long Object)"; }
    String m(Number n) { return "H.m(Number)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`

- (b) `p2.m(42)`

- (c) `h.m(42)`

- (d) `p.m(42.0)`

- (e) `p2.m(42.0)`

- (f) `h.m(42.0)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `int` \leq `Number` e `int` \leq `Long`, nessun metodo di `P` è applicabile per sottotipo

- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing `int` può essere convertito a `Integer`; poiché `Integer` \leq `Number` e `Integer` \leq `Long`, l'unico metodo applicabile è `m(Number)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(Number)` in `P` e viene stampata la stringa "`P.m(Number)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(Number)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(Number)`".

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): i metodi ereditati da `P` non sono applicabili per sottotipo analogamente ai punti precedenti; dei due metodi definiti in `H`, poiché `int` \leq `Long` e `int` \leq `long`, solo `m(long)` è applicabile e viene quindi scelto.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H.m(long)`; viene quindi stampata la stringa "`H.m(long value)`".

(d) Il literal `42L` ha tipo statico `long` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `long` \leq `Number` e `long` \leq `Long`, nessun metodo di `P` è applicabile per sottotipo

- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing `long` può essere convertito a `Long`; poiché `Long` \leq `Number` e `Long` \leq `Long`, entrambi i metodi sono applicabili, ma dato che `Long` \leq `Number` viene scelto `m(Long)` che è il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(Long)` in `P` e viene stampata la stringa "`P.m(Long object)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H` e poiché il metodo con segnatura `m(Long)` è ridefinito in `H`, viene eseguito il corrispondente metodo in `H` e viene stampata la stringa "`H.m(Long object)`".

(f) Il literal `42L` ha tipo statico `long` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): i metodi ereditati da `P` non sono applicabili per sottotipo analogamente ai punti precedenti; dei due metodi definiti in `H`, poiché `long` \leq `Long` e `long` \leq `long`, solo `m(long)` è applicabile e viene quindi scelto.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa "`H.m(long value)`".

giugno 2022

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Object o) { return "P.m(Object)"; }
    String m(Double d) { return "P.m(Double object)"; }
}
public class H extends P {
    String m(Double d) { return "H.m(Double object)"; }
    String m(double d) { return "H.m(double value)"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe Test il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) p.m(42)
- (b) p2.m(42)
- (c) h.m(42)
- (d) p.m(42.0)
- (e) p2.m(42.0)
- (f) h.m(42.0)

febbraio 2022 parziale

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) I literal 42L hanno tipo statico long e il tipo statico di p è P.

- primo tentativo (solo sottotipo): il metodo con solo un parametro non è applicabile, mentre quello con due è accessibile e applicabile per sottotipo, poiché long \leq float; viene scelto l'unico metodo applicabile m(float,float) che è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo m(float,float) in P e viene stampata la stringa "P.m(float,float)".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in p2 è H e poiché il metodo m(float,float) è ridefinito in H, viene eseguito quest'ultimo e, quindi, viene stampata la stringa "H.m(float,float)".

(c) I literal 42L hanno tipo statico long e il tipo statico di h è H.

- primo tentativo (solo sottotipo): H eredita da P solamente il metodo m che non è comunque applicabile dato che ha solo un parametro; i metodi m di H hanno due parametri e sono entrambi accessibili e applicabili per sottotipo, dato che long \leq float e long \leq double; viene scelto il più specifico, ossia m(float,float) poiché float \leq double.

A runtime, il tipo dinamico dell'oggetto in h è H, quindi viene eseguito il metodo m(float,float) di H e viene stampata la stringa "H.m(float,float)".

(d) I literal 42.0 hanno tipo statico double e il tipo statico di p è P.

- primo tentativo (solo sottotipo): il metodo con solo un parametro non è applicabile, ma neanche quello con due, poiché double $\not\leq$ float; viene scelto l'unico metodo applicabile m(float,float) che è ovviamente anche il più specifico.

- secondo e terzo tentativo: anche tali tentativi falliscono poiché non sono applicabili conversioni di tipo boxing/unboxing e non ci sono metodi con aritria variabile, quindi ci sarà un errore di compilazione.

(e) Visto che i tipi statici sono gli stessi, ci sarà un errore di compilazione come nel caso precedente.

(f) I literal 42.0 hanno tipo statico double e il tipo statico di h è H.

- primo tentativo (solo sottotipo): H eredita da P solamente il metodo m che non è comunque applicabile dato che ha solo un parametro; i metodi m di H hanno due parametri, ma l'unico che è sia accessibile sia applicabile per sottotipo è m(double,double) dato che double $\not\leq$ float e double \leq double; viene scelto l'unico metodo applicabile m(double,double) che è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in h è H, quindi viene eseguito il metodo m(double,double) di H e viene stampata la stringa "H.m(double,double)".

gennaio 2022

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) I literal 42.0f hanno tipo statico float e il tipo statico di p è P.

- primo tentativo (solo sottotipo): il metodo con solo un parametro non è applicabile, mentre quello con due è accessibile e applicabile per sottotipo, poiché float \leq float; viene scelto l'unico metodo applicabile m(float,float) che è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo m(float,float) in P e viene stampata la stringa "P.m(float,float)".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in p2 è H, ma poiché il metodo m(float,float) non è ridefinito in H, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "P.m(float,float)".

(c) I literal 42.0f hanno tipo statico float e il tipo statico di h è H.

- primo tentativo (solo sottotipo): i metodi m di H (sia quelli definiti nella classe, sia quelli ereditati da P) con un solo parametro non sono applicabili, mentre quelli con due sono entrambi accessibili e applicabili per sottotipo, dato che float \leq float e float \leq double; viene scelto il più specifico, ossia m(float,float) poiché float \leq double.

A runtime, il tipo dinamico dell'oggetto in h è H, quindi viene eseguito il metodo m(float,float) ereditato da P e viene stampata la stringa "P.m(float,float)".

(d) I literal 42 hanno tipo statico int e il tipo statico di p è P.

- primo tentativo (solo sottotipo): il metodo con solo un parametro non è applicabile, mentre quello con due è accessibile e applicabile per sottotipo, poiché int \leq float; viene scelto l'unico metodo applicabile m(float,float) che è ovviamente anche il più specifico.

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo m(float,float) in P e viene stampata la stringa "P.m(float,float)".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in p2 è H, ma poiché il metodo con segnatura m(float,float) non è ridefinito in H, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "P.m(float,float)".

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal 42 ha tipo statico int e il tipo statico di p è P.

- primo tentativo (solo sottotipo): poiché int $\not\leq$ Object e int $\not\leq$ Double, nessun metodo di P è applicabile per sottotipo
- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing int può essere convertito a Integer; poiché Integer \leq Object e Integer $\not\leq$ Double, l'unico metodo applicabile è m(Object).

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo m(Object) in P e viene stampata la stringa "P.m(Object)".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in p2 è H, ma poiché il metodo m(Object) non è ridefinito in H, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "P.m(Object)".

(c) Il literal 42 ha tipo statico int e il tipo statico di h è H.

- primo tentativo (solo sottotipo): i metodi ereditati da P non sono applicabili per sottotipo analogamente ai punti precedenti; dei due metodi definiti in H, poiché int $\not\leq$ Double e int \leq double, solo m(double) è applicabile e viene quindi scelto.

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo di H m(double); viene quindi stampata la stringa "H.m(double value)".

(d) Il literal 42.0 ha tipo statico double e il tipo statico di p è P.

- primo tentativo (solo sottotipo): poiché double $\not\leq$ Object e double $\not\leq$ Double, nessun metodo di P è applicabile per sottotipo
- secondo tentativo (boxing/unboxing+sottotipo): tramite boxing double può essere convertito a Double; poiché Double \leq Object e Double $\not\leq$ Double, entrambi i metodi sono applicabili, ma dato che Double \leq Object viene scelto m(Double) che è il più specifico.

A runtime, il tipo dinamico dell'oggetto in p è P, quindi viene eseguito il metodo con segnatura m(Double) in P e viene stampata la stringa "P.m(Double object)".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in p2 è H e poiché il metodo con segnatura m(Double) è ridefinito in H, viene eseguito il corrispondente metodo in H e viene stampata la stringa "H.m(Double object)".

(f) Il literal 42.0 ha tipo statico double e il tipo statico di h è H.

- primo tentativo (solo sottotipo): i metodi ereditati da P non sono applicabili per sottotipo analogamente ai punti precedenti; dei due metodi definiti in H, poiché double $\not\leq$ Double e double \leq double, solo m(double) è applicabile e viene quindi scelto.

A runtime, il tipo dinamico dell'oggetto in h è H, quindi per lo stesso motivo del punto precedente viene stampata la stringa "H.m(double value)".

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(float f1, float f2) {return "P.m(float,float)";}
    String m(double[] d) {return "P.m(double[])";}
}
public class H extends P {
    String m(float f1, float f2) {return "H.m(float,float)";}
    String m(double d1, double d2) {return "H.m(double,double)";}
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe Test il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) p.m(42L, 42L)

(b) p2.m(42L, 42L)

(c) h.m(42L, 42L)

(d) p.m(42.0, 42.0)

(e) p2.m(42.0, 42.0)

(f) h.m(42.0, 42.0)

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(float f1, float f2) { return "P.m(float,float)"; }
    String m(double[] f) { return "P.m(double[])"; }
}
public class H extends P {
    String m(double d1, double d2) { return "H.m(double,double)"; }
    String m(double[] d) { return "H.m(double[])"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe Test il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) p.m(42.0f, 42.0f)

(b) p2.m(42.0f, 42.0f)

(c) h.m(42.0f, 42.0f)

(d) p.m(42, 42)

(e) p2.m(42, 42)

(f) h.m(42, 42)

(f) I literal 42 hanno tipo statico int e il tipo statico di h è H.

- primo tentativo (solo sottotipo): i metodi m di H (sia quelli definiti nella classe, sia quelli ereditati da P) con un solo parametro non sono applicabili, mentre quelli con due sono entrambi accessibili e applicabili per sottotipo, dato che int \leq float e int \leq double; viene scelto il più specifico, ossia m(float,float) poiché float \leq double.

A runtime, il tipo dinamico dell'oggetto in h è H, quindi viene eseguito il metodo m(float,float) ereditato da P; viene stampata la stringa "P.m(float,float)".

Settembre 2021

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(double d1, double d2) { return "P.m(double,double)"; }  
    String m(float f1, float f2) { return "P.m(float,float)"; }  
}  
  
public class H extends P {  
    String m(long l1, long l2) { return "H.m(long,long)"; }  
    String m(int i1, int i2) { return "H.m(int,int)"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42, 42)`
- (b) `p2.m(42, 42)`
- (c) `h.m(42, 42)`
- (d) `p.m(42.0, 42.0)`
- (e) `p2.m(42.0, 42.0)`
- (f) `h.m(42.0, 42.0)`

Luglio 2021

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `int ≤ long` e `int ≤ long[]`, viene scelto il metodo `m(long)` di `P` che è l'unico accessibile e applicabile per sottotipo.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa "`P.m(long)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(long)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(long)`".

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché `int ≤ int`, `int ≤ long` e `int ≤ int[]`, `int ≤ long[]`, gli unici metodi accessibili e applicabili per sottotipo sono `m(int)` definito in `H` e `m(long)` ereditato da `P`; viene scelto il metodo più specifico `m(int)` poiché `int ≤ long`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H.m(int)`; viene quindi stampata la stringa "`H.m(int)`".

(d) Il literal `42L` ha tipo `long` e l'espressione `new long[]{42L, 24L}` ha tipo statico `long[]`, mentre il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `long[] ≤ long[]` e `long[] ≤ long`, l'unico metodo di `P` applicabile per sottotipo è `m(long[])`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(long[])` in `P` e viene stampata la stringa "`P.m(long[])`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(long[])` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(long[])`".

(f) Il literal `42L` ha tipo `long` e l'espressione `new long[]{42L, 24L}` ha tipo statico `long[]`, mentre il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché `long[] ≤ int` e `long[] ≤ int[]`, l'unico metodo accessibile e applicabile per sottotipo è `m(long[])` ereditato da `P` come spiegato nei due punti precedenti.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa "`P.m(long[])`".

Giugno 2021

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(double d) { return "P.m(double)"; }  
    String m(float f) { return "P.m(float)"; }  
}  
  
public class H extends P {  
    String m(long l) { return "H.m(long)"; }  
    String m(int i) { return "H.m(int)"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42.0)`
- (e) `p2.m(42.0)`
- (f) `h.m(42.0)`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché entrambi i metodi `m` di `P` hanno due parametri `int ≤ float ≤ double` tutti e due sono accessibili e applicabili per sottotipo; viene scelto il più specifico, ossia `m(float, float)` poiché `float ≤ double`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float, float)` in `P` e viene stampata la stringa "`P.m(float, float)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(float, float)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(float, float)`".

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché tutti i metodi `m` di `H` (sia quelli definiti nella classe, sia quelli ereditati da `P`) hanno due parametri `int ≤ int ≤ long ≤ float ≤ double` tutti sono accessibili e applicabili per sottotipo; viene scelto il più specifico, ossia `m(int, int)` poiché `int ≤ long ≤ float ≤ double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H.m(int, int)`; viene quindi stampata la stringa "`H.m(int, int)`".

(d) Il literal `42.0` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `double ≤ double` e `double ≤ float`, l'unico metodo di `P` applicabile per sottotipo è `m(double, double)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(double, double)` in `P` e viene stampata la stringa "`P.m(double, double)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(double, double)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(double, double)`".

(f) Il literal `42.0` ha tipo `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): tra i quattro metodi `m` di `H` (sia quelli definiti nella classe, sia quelli ereditati da `P`) l'unico accessibile e applicabile per sottotipo è `m(double, double)`, dato che `double ≤ double`, `double ≤ int`, `double ≤ long` e `double ≤ float`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa "`P.m(double, double)`".

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {  
    String m(long l) { return "P.m(long)"; }  
    String m(long[] la) { return "P.m(long[])"; }  
}  
  
public class H extends P {  
    String m(int i) { return "H.m(int)"; }  
    String m(int[] ia) { return "H.m(int[])"; }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        P p = new P();  
        H h = new H();  
        P p2 = h;  
        System.out.println(...);  
    }  
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p.m(42)`

(b) `p2.m(42)`

(c) `h.m(42)`

(d) `p.m(new long[]{42L, 24L})`

(e) `p2.m(new long[]{42L, 24L})`

(f) `h.m(new long[]{42L, 24L})`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `int ≤ double` e `int ≤ float`, entrambi i metodi di `P` sono accessibili e applicabili per sottotipo, ma viene scelto il metodo più specifico `m(float)` poiché `float ≤ double`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(float)` in `P` e viene stampata la stringa "`P.m(float)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(float)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(float)`".

(c) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): sia i metodi ereditati da `P`, sia quelli definiti in `H` sono applicabili per sottotipo dato che `int ≤ double`, `int ≤ float`, `int ≤ long` e `int ≤ long[]`, ma viene scelto il metodo più specifico `m(int)` poiché `int ≤ long ≤ float ≤ double`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H.m(int)`; viene quindi stampata la stringa "`H.m(int)`".

(d) Il literal `42.0` ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `double ≤ double` e `double ≤ float`, l'unico metodo di `P` applicabile per sottotipo è `m(double, double)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(double, double)` in `P` e viene stampata la stringa "`P.m(double, double)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(double, double)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(double, double)`".

(f) Il literal `42.0` ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché `double ≤ long` e `double ≤ int`, l'unico metodo applicabile per sottotipo è `m(double)` ereditato da `P` come spiegato nei due punti precedenti.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa "`P.m(double)`".

giugno 2021

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Short s) { return "P.m(Short)"; }
    String m(Number n) { return "P.m(Number)"; }
}

public class H extends P {
    String m(short s) { return "H.m(short)"; }
    String m(float f) { return "H.m(float)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `h.m(42)`
- (d) `p.m(42.0)`
- (e) `p2.m(42.0)`
- (f) `h.m(42.0)`

febbraio 2020

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il tipo statico di `p` è `P` e quello di `f` è `float`.

- prima fase (applicabilità per sottotipo): poiché `float` \leq `double` e `float` $\not\leq$ `double[]`, solo il metodo `m(double)` di `P` è applicabile.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double)` in `P` e viene stampata la stringa "`P.m(double)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto (a), visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(double)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto (a) e, quindi, viene stampata la stringa "`P.m(double)`".

(c) Il tipo statico di `h` è `H` mentre l'argomento `f` ha sempre tipo statico `float`.

- prima fase (applicabilità per sottotipo): oltre al metodo `m(double)` ereditato da `P` è applicabile per le stesse ragioni dei punti (a) e (b), risulta anche applicabile il metodo `m(float)` di `H` (dato che ovviamente `float` \leq `float`), mentre non lo è `m(float[])` poiché `float` $\not\leq$ `float[]`; poiché `float` \leq `double` viene selezionato il metodo più specifico `m(float)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(float)` di `H`; poiché il parametro `f` ha tipo statico `float` e `super` si riferisce alla classe `P`, la chiamata `super.m(f)` si comporta come al punto (a) e quindi viene stampata la stringa "`P.m(double)` `H.m(float)`".

(d) Il tipo statico di `p` è `P` e quello di `da` è `double[]`.

- prima fase (applicabilità per sottotipo): poiché `double[]` \leq `double[]` e `double[]` $\not\leq$ `double`, solo il metodo `m(double[])` di `P` è applicabile.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(double[])` in `P` e viene stampata la stringa "`P.m(double[])`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto (d), visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(double[])` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto (d) e viene stampata la stringa "`P.m(double[])`".

(f) Il tipo statico di `h` è `H` mentre l'argomento `da` ha sempre tipo statico `double[]`.

- prima fase (applicabilità per sottotipo): oltre al metodo `m(double[])` ereditato da `P` è applicabile per le stesse ragioni dei punti (d) ed (e), non sono applicabili altri metodi poiché `double[]` $\not\leq$ `float` e `double[]` $\not\leq$ `float[]`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, ma poiché il metodo `m(double[])` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto (d) e viene stampata la stringa "`P.m(double[])`".

gennaio 2020

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, tutti i metodi sono accessibili.

(a) Il tipo statico di `p` è `P`, il literal 42 ha tipo statico `int`.

- prima fase (applicabilità per sottotipo): poiché `int` \leq `long` e `int` $\not\leq$ `long[]` (nella prima e seconda fase `long...` viene considerato uguale al tipo `array long[]`), solo il metodo `m(long)` di `P` è applicabile.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long)` in `P` e viene stampata la stringa "`P.m(long)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto (a), visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(long)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto (a) e, quindi, viene stampata la stringa "`P.m(long)`".

(c) Il tipo statico di `h` è `H` mentre l'argomento `i` ha sempre tipo statico `int`.

- prima fase (applicabilità per sottotipo): oltre al metodo `m(long)` ereditato da `P` è applicabile per le stesse ragioni dei punti (a) e (b), risulta anche applicabile il metodo `m(int)` di `H` (dato che ovviamente `int` \leq `int`), mentre non lo è `m(int...)` poiché `int` $\not\leq$ `int...` (nella prima e seconda fase `int...` viene considerato uguale al tipo `array int[]`); poiché `int` \leq `long` viene selezionato il metodo più specifico `m(int)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int)` di `H`; poiché il parametro `i` ha tipo statico `int` e `super` si riferisce alla classe `P`, la chiamata `super.m(i)` si comporta come al punto (a); viene quindi stampata la stringa "`P.m(long)` `H.m(int)`".

(d) Il literal 42 ha tipo statico `int` e il tipo statico di `h` è `P`.

- prima fase (applicabilità per sottotipo): poiché entrambi i metodi `m` di `P` hanno un solo parametro (il metodo `m(long...)` è considerato con numero variabile di parametri solo nella terza fase) mentre gli argomenti sono due, nessun metodo è applicabile;

- seconda fase (applicabilità per boxing/unboxing e sottotipo): nessun metodo `m` di `P` è applicabile per le stesse ragioni del punto precedente;

- terza fase (applicabilità per arietà variabile, boxing/unboxing e sottotipo): il metodo `m(long...)` viene considerato con un numero variabile di parametri di tipo `long` ed è quindi applicabile dato che `int` \leq `long`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `m(long...)` in `P` e viene stampata la stringa "`P.m(long...)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto (d), visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo `m(long...)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto (d) e viene stampata la stringa "`P.m(long...)`".

(f) Il literal 42 ha tipo statico `int` e il tipo statico di `h` è `H`.

- prima fase (applicabilità per sottotipo): per le stesse ragioni dei punti (d) ed (e) non sono applicabili né i metodi di `P`, né quelli di `H`;

- seconda fase (applicabilità per boxing/unboxing e sottotipo): nessun metodo `m` di `P` e `H` è applicabile per le stesse ragioni del punto precedente;

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal 42 ha tipo statico `int` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `int` $\not\leq$ `Short` e `int` $\not\leq `Number`, non esistono metodi di `P` accessibili e applicabili per sottotipo;$
- secondo tentativo (boxing/unboxing e sottotipo): `int` può essere convertito a `Integer` mediante boxing; poiché `Integer` $\not\leq$ `Short` e `Integer` \leq `Number`, solo il metodo `m(Number)` di `P` è applicabile per boxing e reference widening.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(Number)` in `P` e viene stampata la stringa "`P.m(Number)`".

(b) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(Number)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(Number)`".

(c) Il literal 42 ha tipo statico `int` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): tra i metodi ereditati da `P` e quelli definiti in `H`, solo `m(float)` è applicabile per sottotipo dato che `int` \leq `float`, ma `int` $\not\leq$ `short`, `int` $\not\leq$ `Number`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo di `H` con segnatura `m(float)`; viene quindi stampata la stringa "`H.m(float)`".

(d) Il literal 42.0 ha tipo statico `double` e il tipo statico di `p` è `P`.

- primo tentativo (solo sottotipo): poiché `double` $\not\leq$ `Short` e `double` $\not\leq$ `Number`, non esistono metodi di `P` accessibili e applicabili per sottotipo;
- secondo tentativo (boxing/unboxing e sottotipo): `double` può essere convertito a `Double` mediante boxing e `Double` \leq `Number`, ma `Double` $\not\leq$ `Short`, `Double` $\not\leq$ `float`; quindi, tra i metodi ereditati da `P` e quelli definiti in `H`, solo il metodo `m(Number)` di `P` è applicabile per boxing e reference widening.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(Number)` in `P` e viene stampata la stringa "`P.m(Number)`".

(e) L'espressione è staticamente corretta e l'overloading viene risolto come al punto precedente, visto che i tipi statici sono gli stessi.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, ma poiché il metodo con segnatura `m(Number)` non è ridefinito in `H`, viene eseguito lo stesso metodo del punto precedente e, quindi, viene stampata la stringa "`P.m(Number)`".

(f) Il literal 42.0 ha tipo statico `double` e il tipo statico di `h` è `H`.

- primo tentativo (solo sottotipo): poiché `double` $\not\leq$ `Short`, `double` $\not\leq$ `Number`, `double` $\not\leq$ `short`, `double` $\not\leq$ `float`, non esistono metodi ereditati da `P` o definiti in `H` accessibili e applicabili per sottotipo;

- secondo tentativo (boxing/unboxing e sottotipo): `double` può essere convertito a `Double` mediante boxing e `Double` \leq `Number`, ma `Double` $\not\leq$ `Short`, `Double` $\not\leq$ `float`; quindi, tra i metodi ereditati da `P` e quelli definiti in `H`, solo il metodo `m(Number)` di `P` è applicabile per boxing e reference widening.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi per lo stesso motivo del punto precedente viene stampata la stringa "`P.m(Number)`".

```
public class P {
    String m(double d) {
        return "P.m(double)";
    }
    String m(double[] da) {
        return "P.m(double[])";
    }
}
public class H extends P {
    String m(float f) {
        return super.m(f) + " H.m(float)";
    }
    String m(float[] fa) {
        return super.m(new double[]{fa[0], fa[1]}) + " H.m(float[])";
    }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        float f = 4.2f;
        double[] da = { 1.2, 2.3 };
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(f)` (b) `p2.m(f)` (c) `h.m(f)` (d) `p.m(da)` (e) `p2.m(da)` (f) `h.m(da)`

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Long i) {return "P.m(long)";}
    String m(Long... i) {return "P.m(long...)";}
}
public class H extends P {
    String m(int i) {return super.m(i) + " H.m(int);">
    String m(int... i) {return super.m(i[0], i[1]) + " H.m(int...)";}
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)` (b) `p2.m(42)` (c) `h.m(42)` (d) `p.m(42,42)` (e) `p2.m(42,42)` (f) `h.m(42,42)`

(x) • terza fase (applicabilità per arietà variabile, boxing/unboxing e sottotipo): oltre al metodo `m(long...)` ereditato da `P` e applicabile per le stesse ragioni dei punti (d) ed (e), il metodo `m(int...)` in `H` viene considerato con un numero variabile di parametri di tipo `int` ed è quindi applicabile (dato che ovviamente `int` \leq `int`); poiché `int` \leq `long` viene selezionato il metodo più specifico `m(int...)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `m(int...)` di `H`; poiché il parametro `i` ha tipo statico `int[]`, `super` si riferisce alla classe `P` e gli argomenti `i[0]` e `i[1]` hanno tipo `int`, la chiamata `super.m(i[0], i[1])` si comporta come al punto (d); viene quindi stampata la stringa "`P.m(long...)` `H.m(int...)`".

Gennaio 2016

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(int i) { return "P.m(int)"; }
    String m(long l) { return "P.m(long)"; }
}

public class H extends P {
    String m(int i) { return super.m(i) + " H.m(int)"; }
    String m(Integer i) { return super.m(i) + " H.m(Integer)"; }
    String m(Integer... l) { return (l.length > 0 ? super.m(l[0]) : "") + " H.m(Integer...)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m(42)`
- (c) `p.m(new Long(42))`
- (d) `h.m(new Long(42))`
- (e) `p2.m(42, 42)`
- (f) `h.m(42, 42)`

Settembre 2015

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal '`a'` ha tipo statico `char`; il tipo statico di `p` è `P` ed esiste un solo metodo di `P` accessibile e applicabile per sotto-tipo, `String m(char c)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `String m(char c)` in `P`.

Viene stampata la stringa "`P.m(char)`".

(b) L'espressione è staticamente corretta per esattamente lo stesso motivo del punto precedente, visto che `p2` ha lo stesso tipo statico di `p`.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, quindi viene eseguito il metodo `String m(char c)` in `H`. Poiché `c` ha tipo statico `char`, per l'invocazione `super.m(c)` esiste un solo metodo in `P` accessibile e applicabile per sotto-tipo, `String m(char c)`.

Viene stampata la stringa "`P.m(char) H.m(char)`".

(c) Il literal '`a'` ha tipo statico `char` e per l'invocazione `Character.valueOf('a')` esiste un solo metodo statico nella classe `Character` accessibile e applicabile per sotto-tipo, che restituisce un valore di tipo `Character`. Il tipo statico di `i` è `H` ed esiste un solo metodo di `H` accessibile e applicabile per sotto-tipo, `String m(Character c)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `String m(Character c)` in `H`.

Poiché `c` ha tipo statico `Character`, per l'invocazione `super.m(c)` non esistono metodi in `P` accessibili e applicabili per sotto-tipo, mentre `String m(char c)` è l'unico accessibile e applicabile per unboxing.

Viene stampata la stringa "`P.m(char) H.m(Character)`".

(d) Il literal "`a`" ha tipo statico `String`; il tipo statico di `p` è `P` ed esiste un solo metodo di `P` accessibile e applicabile per sotto-tipo, `String m(String s)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `String m(String s)` in `P`.

Viene stampata la stringa "`P.m(String)`".

(e) L'espressione `new char[]{'4', '2'}` ha tipo statico `char[]`, il tipo statico di `p2` è `P` e non esiste alcun metodo di `P` accessibile e applicabile, quindi viene segnalato un errore durante la compilazione.

(f) L'espressione `new Character[]{'4', '2'}` ha tipo statico `Character[]`, il tipo statico di `h` è `H` e l'unico metodo di `H` che è accessibile e applicabile è `String m(Character... cs)`, poiché `Character...` corrisponde a `Character[]`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `String m(Character... cs)` in `H`. La variabile `sb` viene inizializzata con un'istanza di `StringBuilder` corrispondente alla stringa vuota. In seguito, gli elementi dell'array vengono inseriti nell'istanza di `StringBuilder` nell'ordine e separati da uno spazio bianco, dopo di che viene concatenata in fondo la stringa "`H.m(Character...)`".

Viene stampata la stringa "`4 2 H.m(Character...)`".

Luglio 2015

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(byte b) { return "P.m(byte)"; }
    String m(int i) { return "P.m(int)"; }
    String m(Number... n) { return "P.m(Number...)"; }
}

public class H extends P {
    String m(float f) { return super.m(f) + " H.m(float)"; }
    String m(short s) { return super.m(s) + " H.m(short)"; }
    String m(Double... ds) { return super.m(ds) + " H.m(Double...)"; }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m(42)`
- (b) `p2.m((byte) 42)`
- (c) `h.m((float) 42)`
- (d) `p.m(Short.valueOf((byte) 42))`
- (e) `p2.m(new Double[] { 4.2 })`
- (f) `h.m(new double[]{4.2})`

Soluzione: assumendo che tutte le classi siano dichiarate nello stesso package, si hanno i seguenti casi:

(a) Il literal `42` ha tipo statico `int`; il tipo statico di `p` è `P` ed entrambi i metodi di `P` sono accessibili e applicabili per sotto-tipo, ma quello con segnatura `m(int)` è più specifico.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(int)` in `P`. Viene stampata la stringa "`P.m(int)`".

(b) L'espressione è staticamente corretta per esattamente lo stesso motivo del punto precedente, visto che `p2` ha lo stesso tipo statico di `p`.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, quindi viene eseguito il metodo con segnatura `m(int)` ridefinito in `H`. Poiché `i` ha tipo statico `int`, l'overloading per l'invocazione `super.m(i)` viene risolta come al punto precedente e, quindi, viene chiamato il metodo della classe `P` con segnatura `m(int)`.

Viene stampata la stringa "`P.m(int) H.m(int)`".

(c) L'espressione `new Long(42)` ha tipo statico `Long` poiché `42` ha tipo `int` che è sotto-tipo del tipo `long` del parametro dell'unico costruttore applicabile per sotto-tipo di `Long` (l'altro ha tipo `String`); il tipo statico di `p` è `P`. Nessun metodo della classe `P` è applicabile per sotto-tipo, mentre per unboxing l'unico metodo accessibile e applicabile è quello con segnatura `m(long)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo con segnatura `m(long)` in `P`. Viene stampata la stringa "`P.m(long)`".

(d) L'espressione `new Long(42)` ha tipo statico `Long` per gli stessi motivi del punto precedente e il tipo statico di `h` è `H`; nessun metodo in `H` è applicabile per sotto-tipo o per unboxing, quindi l'overloading viene risolto come al punto precedente.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo in `P` con segnatura "`P.m(long)`". Viene stampata la stringa "`P.m(long)`".

(e) Il literal `42` ha tipo statico `int`; il tipo statico di `p2` è `P` e nessun metodo accessibile di `P` è applicabile visto che entrambi i metodi hanno arità costante 1, quindi verrà segnalato un errore di tipo.

(f) Il literal `42` ha tipo statico `int` e il tipo statico di `h` è `H`; nessun metodo di `H` è applicabile per sotto-tipo o per boxing, dato che in entrambi i casi tutti i metodi considerati hanno arità costante 1. Il metodo in `H` con arità variabile e segnatura `m(Integer...)` è applicabile visto che `int` può essere convertito a `Integer` tramite boxing.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo in `H` con segnatura `m(Integer...)`. Poiché in questo caso vengono passati due argomenti `l.length` si valuta in `true` e viene eseguita l'invocazione `super.m(l[0])`; l'argomento `l[0]` ha tipo statico `Integer`, nessun metodo accessibile in `P` è applicabile per sotto-tipo, mentre entrambi i metodi sono applicabili per unboxing, ma quello con segnatura `m(int)` è più specifico.

Viene stampata la stringa "`P.m(int) H.m(Integer...)`".

```
public class P {
    String m(char c) { return "P.m(char)"; }
    String m(String s) { return "P.m(String)"; }
}

public class H extends P {
    String m(char c) { return super.m(c) + " H.m(char)"; }
    String m(Character c) { return super.m(c) + " H.m(Character)"; }
    String m(Character... cs) {
        StringBuilder sb = new StringBuilder();
        for (Character c : cs)
            sb.append(c).append(" ");
        return sb.append("H.m(Character...)").toString();
    }
}

public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

- (a) `p.m('a')`
- (b) `p2.m('a')`
- (c) `h.m(Character.valueOf('a'))`
- (d) `p.m("a")`
- (e) `p2.m(new char[] { '4', '2' })`
- (f) `h.m(new Character[] { '4', '2' })`

Soluzione:

(a) Il literal `42` ha tipo statico `int`; il tipo statico di `p` è `P` ed esiste un solo metodo di `P` accessibile e applicabile per sotto-tipo, `String m(int i)`.

A runtime, il tipo dinamico dell'oggetto in `p` è `P`, quindi viene eseguito il metodo `String m(int i)` in `P`.

Viene stampata la stringa "`P.m(int)`".

(b) Il literal `42` ha tipo statico `int`, il cast a `byte` risulta staticamente corretto e il tipo statico dell'argomento è `byte`; il tipo statico di `p2` è `P` ed esistono due metodi di `P` accessibili e applicabili per sotto-tipo: `String m(byte b)` e `String m(int i)`. Viene selezionato il primo metodo, poiché è il più specifico.

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, quindi il metodo `String m(byte b)` viene cercato a partire da `H`, ma poiché il metodo non viene ridefinito viene eseguito quello di `P`. La conversione di tipo dovuta al cast non comporta in questo caso perdita di informazione, dato che la costante `42` è correttamente rappresentabile con un valore di tipo `byte`.

Viene stampata la stringa "`P.m(byte)`".

(c) Il literal `42` ha tipo statico `double`, il cast a `float` risulta staticamente corretto e il tipo statico dell'argomento è `float`; il tipo statico di `h` è `H` ed esiste un solo metodo di `H` accessibile e applicabile per sotto-tipo, `String m(float f)`.

A runtime, il tipo dinamico dell'oggetto in `h` è `H`, quindi viene eseguito il metodo `String m(float f)` in `H`. La conversione di tipo dovuta al cast non comporta in questo caso perdita di informazione, dato che la costante `42` è correttamente rappresentabile con un valore di tipo `float`.

Poiché il parametro `f` ha tipo statico `float`, per la chiamata `super.m(f)` non esiste alcun metodo in `P` di arità costante accessibile e applicabile per sotto-tipo o per conversione di tipo boxing/unboxing, mentre il metodo di arità variabile è accessibile e applicabile per boxing da `float` a `Float` e reference widening da `Float` a `Number`.

Viene stampata la stringa "`P.m(Number...) H.m(float)`".

(d) L'argomento di `Short.valueOf` ha tipo statico `byte` come al punto (b) ed esiste un solo metodo statico nella classe `Short` accessibile e applicabile per sotto-tipo, che restituisce un valore di tipo `Short`. Il tipo statico di `p` è `P` ed esiste un solo metodo di `P` accessibile e applicabile per sotto-tipo, `String m(int i)`.

Viene stampata la stringa "`P.m(int)`".

(e) Il literal `42` ha tipo statico `double`, l'argomento `new Double[]{4.2}` ha tipo statico `Double[]` (4.2 viene implicitamente convertito tramite boxing) e il tipo statico di `p2` è `P`; esiste un solo metodo di `P` accessibile e applicabile per sotto-tipo, `String m(Number... n)` (durante il controllo di applicabilità per sotto-tipo il parametro `n` viene considerato di tipo `Number[]` e `Double[]` ≤ `Number[]` poiché `Double` ≤ `Number`).

A runtime, il tipo dinamico dell'oggetto in `p2` è `H`, quindi il metodo `String m(byte b)` viene cercato a partire da `H`, ma poiché il metodo non viene ridefinito viene eseguito quello di `P`.

Viene stampata la stringa "`P.m(Number...)`".

(f) Il literal `42` ha tipo statico `double`, l'argomento `new double[]{4.2}` ha tipo statico `double[]` e il tipo statico di `h` è `H`; poiché `double[]` ≤ `Number[]` non esistono metodi accessibili e applicabili, quindi la chiamata non è staticamente corretta.