Esercizio 1

Considerate la seguente gerarchia di classi:

```
interface M { void m(double x); }
interface N { void n(double x); }

class A implements M {
  public void m(double x) { System.out.println("A.m(double)"); }
}

class B extends A {
  public void m(double x) { System.out.println("B.m(double)"); }
}

class C extends A implements N {
  public void n(double x) { System.out.println("C.n(double)"); }
  public void m(int x) { System.out.println("C.m(int)"); }
}
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti?

1. `N x = new C(); ((M)x).m(1);`

2. `C x = new B(); ((B)x).m(1);`

3. `N x = new B(); x.m(1);`

4. `M x = new C(); ((C)x).m(1);`

Soluzione

1. `N x = new C();` tipo statico `N`, tipo dinamico `C`
   `((M)x)` cast ha successo (ha tipo dinamico `C` che è sottotipo di `M`)
   `((M)x).m(1.0)` tipo statico `M`, best match {`m(double)`}, tipo dinamico `C`, implementazione selezionata (override) `C.m(double)`
   risultato: `A.m(int)`

2. `C x = new B();` errore in compilazione: `B` non è un sottotipo di `C`

3. `N x = new B();` errore in compilazione: `B` non è un sottotipo di `N`

4. `M x = new C();` tipo statico `M`, tipo dinamico `C`
   `((C)x)` cast ha successo (ha tipo dinamico `C`)
   `((C)x).m(1.0)` tipo statico `C`, best match {`C.m(int)`}
   risultato: `C.m(int)`

Esercizio 2

Considerate la seguente gerarchia di classi:

```
interface M { M m(); }
interface N { }
class A implements M {
  public M m() { return new B(); }
}
class B extends A {
  public M m() { return new A(); }
}
class C extends A implements N {
  public M m() { return this; }
}
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti?

1. `N x = new C(); M y = x.m()`

2. `M x = new A(); B y = (B)x.m();`

3. `M x = new B(); B y = (B)x.m();`

Soluzione

1. `N x = new C();` tipo statico `N`, tipo dinamico `C`, sottotipo di `N`.
   `x.m()` errore di compilazione: `N` non ha il metodo `m()`.

2. `M x = new A();` tipo statico `M`, tipo dinamico `A`, sottotipo di `M`.
   `x.m()` chiama `A.m()`: tipo statico `M`, tipo dinamico `B`.
   `(B)x.m();` cast valido (tipo dinamico `B`), e assegnazione a `B y` valida.

3. `M x = new B();` tipo statico `M`, tipo dinamico `B`.
   `x.m()` chiama `B.m()`: tipo statico `M`, tipo dinamico `A`.
   `(B)x.m();` cast non valido: tipo dinamico `A` non Ã¨ un sottotipo di `B`. Eccezione a runtime.

Esercizio 3

Considerate la seguente gerarchia di classi:

```
class A {
public void print(String s) { System.out.println(s); }
public void m1() { print("A.m1"); m2(); }
public void m2() { print("A.m2"); }
}
class B extends A {
public void m2() { print("B.m2"); }
public void m3() { print("B.m3"); }
}
class C extends A {
public void m1() { print("C.m1"); }
public void m2() { print("C.m2"); m1(); }
}
class D extends C {
public void m1() { super.m1(); print("D.m1"); }
public void m3() { print("D.m3"); }
}
```

e assumete le seguenti dichiarazioni di variabile.

```
A var1 = new B();  A var2 = new D();
B var3 = new B();  C var4 = new C();
C var5 = new D();  Object var6 = new C();
```

Indicare l'output prodotto dalle seguenti espressioni. Se il comando produce più di una linea di output, utilizzate il carattere '/' per indicare le diverse linee. Se il comando causa errore, indicate il tipo di errore (compilazione o run-time).

1. `var1.m1();`
2. `var2.m1();`
3. `var3.m1();`
4. `var4.m1();`
5. `var5.m1();`
6. `var6.m1();`
7. `((B)var1).m3();`
8. `((B)var2).m3();`
9. `((D)var4).m3();`
10. `(D)var5).m3();`
11. `((D)var6).m3();`

Soluzione

1. `var1.m1();` `A.m1 / B.m2`
2. `var2.m1();` `C.m1 / D.m1`
3. `var3.m1();` `A.m1 / B.m2`
4. `var4.m1();` `C.m1`
5. `var5.m1();` `C.m1 / D.m1`
6. `var6.m1();` compiler error
7. `((B)var1).m3();` `B.m3`
8. `((B)var2).m3();` runtime error
9. `((D)var4).m3();` runtime error
10. `(D)var5).m3();` `D.m3`
11. `((D)var6).m3();` runtime error

Esercizio 4

Considerate la seguente gerarchia di classi:

```
class A {
public void print(String s) { System.out.println(s);public void m(int i)
 { print("A.m(int)"); }
public void m(boolean b) { print("A.m(boolean)"); }
}
public A k() { return this; }
}
class B extends A {
public void m(float f) { print("B.m(float)"); }
public void m(boolean b) { print("B.m(boolean)"); }
}
class C extends A {
public void m(int i) { print("C.m(int)"); }
public void m(double d) { print("C.m(double)"); }
}
```

e assumete le seguenti dichiarazioni di variabile.

```
A va = new A();  A vab = new B();
B vb = new B();  A vac = new C();
C vc = new C();
```

Indicare l'output prodotto dalle seguenti espressioni. Se il comando produce più di una linea di output, utilizzate il carattere '/' per indicare le diverse linee. Se il comando causa errore, indicate il tipo di errore (compilazione o run-time).

1. `vab.m(1);`
2. `vab.m(1.2);`
3. `vac.m(1);`
4. `vb.m(1.2);`
5. `((B)vab).m(1.2);`
6. `((B)(vab.k())).m(1);`
7. `((B)(va.k())).m(1);`
8. `((C)(vac.k())).m(false);`
9. `((B)vb.k()).m(true);`
10. `vac.k().m(1.2);`

Soluzione

1. `vab.m(1);` `A.m(int)`
2. `vab.m(1.2);` compiler error
3. `vac.m(1);` `C.m(int)`
4. `vb.m(1.2);` `B.m(float)`
5. `((B)vab).m(1.2);` `B.m(float)`
6. `((B)(vab.k())).m(1);` `A.m(int)`
7. `((B)(va.k())).m(1);` runtime error
8. `((C)(vac.k())).m(false);` `A.m(boolean)`
9. `((B)vb.k()).m(true);` `B.m(boolean)`
10. `vac.k().m(1.2);` compiler error