Ingegneria del Software

Esercitazione 2

Valutazione Parametri

Illustrare e motivare il valore delle variabili i, counter e counter 2 ad ogni riga del main

```
class Counter {

  int counter = 0;

  public void increment(){
      counter++;
  }

  public void incrementAndSet(int i){
      i++;
      counter=i;
  }

  public void incrementAndSet(Counter c){
       c.counter++;
       counter = c.counter
  }
}
```

```
public static void main(String args[]) {
    Counter counter = new Counter();
    counter.increment();
    int i = 3;
    counter.incrementAndSet(i);
    Counter counter2 = new Counter();
    counter2.incrementAndSet(i);
    counter.incrementAndSet(counter2);
    Counter counter3 = counter;
  }
```

Valutazione Parametri

Tipi primitivi: passati per valore
Oggetti: passati per referenza

Risposta:

```
public static void main(String args[]) {
    Counter counter = new Counter();          counter = 0
    counter.increment();                      counter = 1
    int i = 3;                                i = 3
    counter.incrementAndSet(i);               i = 3, counter = 4
    Counter counter2 = new Counter();         i = 3, counter = 4, counter2 = 0
    counter2.incrementAndSet(i);              i = 3, counter = 4, counter2 = 4
    counter.incrementAndSet(counter2);        i = 3, counter = 5, counter2 = 5
}
```

Strings

Cosa stampa questo programma?

```
public class StringDemo {
 public static void main(String[] args){
  String s1 = “Guess who”;
  String s2 = s1;
  String s3 = “ABC”;
  s1 = s1 + “ is back”;
  s1 = s3;
  System.out.println(s1);
  System.out.println(s2);
 }
}
```

Strings are Immutable

Risposta:

> Guess who is back?

> Guess who

- `s2` punta a `s1`
- `s1` concatenato con un'altra stringa genera un nuovo oggetto
- `s2` continua a puntare all'oggetto precedente

Persons and Students

Definire le classi Person, Student e Grade

Specifiche:

- Una persona ha un nome, cognome e una data *(java.util.Date)*
- Uno studente è una persona con un id e una lista di voti
- Un voto contiene punteggio e crediti
- Lo studente espone due funzionalità:
  - Calcolo media pesata
  - Controllo se è possibile che si laurei (crediti totali >= 180)

Access Modifier

Completare il codice con gli opportuni modificatori di visibilità in modo tale che l'accesso alle variabili e ai metodi sia il più ristretto possibile ma che non crei errori di compilazione

Memento: Access Modifier

`public`	visibile da qualsiasi parte del programma
`private`	visibile solo dall'interno della classe stessa
`protected`	visibile solo dalle classi dello stesso package e dalle sottoclassi
`default`	visibile dallo stesso package e dalle sottoclassi se sono nello stesso pacchetto.

```
package a;
... class First {
  ... int x;
  ... int y;
  ... void h() { y = -1; }
}
... class Second extends First {
  ... void f(int x) { this.x = x; h(); }
}

package b;
imports a.*;
class Third {
  public static void main(String[] s) {
    Second z = new Second();
    z.f(3);
}

class Fourth extends First {   void g() { h(); }  }
```

```
package a;
public class First {
  int x;  // default
  private int y;
  protected void h() { y = -1; }
}
public class Second extends First {
  public void f(int x) { this.x = x; h(); }
}

package b;
imports a.*;
class Third {
  public static void main(String[] s) {
    Second z = new Second();
    z.f(3);
}
class Fourth extends First {   void g(void) { h(); }  }
```

Access Modifier (2)

Questo codice è corretto?
Che output produce una chiamata al metodo m1 di C2?

```
package A;
public class C1 {
    public void m1() { System.out.print(“ 1”); }
    protected void m2() { System.out.print(“ 2”); }
    private void m3() { System.out.print(“ 3”); }
}

package B;
import A.*;
public class C2 extends C1 {
    public void m1() { System.out.print("4"); m2(); m3(); }
    protected void m2() { System.out.print(“ 5”); }
    private void m3() { System.out.print(“6"); }
}
```

Access Modifier (2)

Risposta:

È corretta.

Una chiamata a m1 di C2 stampa “456"

C2 non vede la definizione di *m3* data da *C1*, perché questa è *private*. Pertanto la definizione di *m3* in *C2* è, banalmente, la definizione di un nuovo metodo. Inoltre *C2* ridefinisce i metodi *m1* e *m2* ereditati da *C1*.

Access Modifier (2)

E questa definizione di C2 è corretta? Perché?

```
public class C2 extends C1 {
    public void m1() {
      System.out.print("4");
      m2();
      m3();
   }
    protected void m2() {
      System.out.print(“ 5”);
    }
}
```

Access Modifier (2)

Risposta:

È scorretta.

Il metodo m3 è definito private nella classe C1 e pertanto non è visibile in C2. Si ottiene quindi un errore a compile-time.

Access Modifier (2)

E questa definizione di C2 è corretta? Perché?

```
public class C2 extends C1 {
    public void m1() {
      System.out.print("4");
      m2();
      m3();
   }
    private void m3() {
      System.out.print(“6”);
    }
}
```

Access Modifier (2)

Risposta:

È corretta.

Il metodo m3 è definito nella classe C2 e il metodo m2 è definito protected in C1 quindi visibile in C2. Una chiamata ad m1 stampa “4 2 6”

Static vs Dynamic Types

Sia dato il seguente frammento di codice.

Indicare gli errori a compile-time.

Eliminare le istruzioni che generano errore a compile-time, e dire se il codice genera errori a runtime.

Eliminare anche le istruzioni che generano errore a runtime, e dire cosa produce in output il programma.

Static vs Dynamic Types

```
package C;

public class C3 {
  public static void main(String[] s) {
    C1 c1;    C2 c2;  Object o;
    c1 = new C1();  /*1*/
    c1.m1();        /*2*/
    c2 = new C2();  /*3 */
    c2.m2();        /*4 */
    c1 = c2;        /*5 */
    c1.m1();        /*6 */
    c2 = new C1();  /*7 */
    o = new C1();   /*8 */
    c2 = (C2) o;    /*9 */
    o = new C2();   /*10 */
   c1 = (C1) o;    /*11 */
    c1.m1();        /*12 */
  }
}
```

```
package A;
public class C1 {
    public void m1() { // 1 }
    protected void m2() { // 2 }
    private void m3() { // 3 }
}

package B;
import A.*;
public class C2 extends C1 {
    public void m1() { // 4,5,6 }
    protected void m2() { // 5 }
    private void m3() { // 6 }
}
```

Static vs Dynamic Types

Risposta:

- **1**, **2**, **3** sono **corrette**. *Il costruttore di default non è definito nella classe, ma dal momento che nessun altro costruttore è definito può comunque essere usato. Il metodo m1 è public e quindi può essere usato da chi importa il package, quindi 3 è corretta produce in output "1", essendo C1 il tipo dinamico di c1.*
- **4** è **scorretta**. *Il metodo m2 è protected e C3 non è nello stesso package di C2 e non è neanche una sottoclasse di C2.*
- **5** e **6** sono **corrette**. *c2 conteneva un oggetto valido, e genera in output "456" essendo C2 il tipo dinamico.*
- **7** è **scorretta**. *Si assegna a c2 un oggetto il cui tipo dinamico è un sovra-tipo.*
- **8** e **10** sono **corrette.** C1 e C2 sono sottotipi di Object.
- **9** è **corretta** ma **genera un errore runtime.** *Il casting non può avere successo perché la variabile o, a runtime, riferisce un oggetto il cui tipo dinamico è C1, che è un sovra-tipo di C2, il tipo che viene indicato nell'operatore di casting.*
- **11** e **12** sono **corrette**. *L'ultima riga produce in output "456".*