

Ingegneria del Software

Esercitazione 3

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() { }
    protected void m2() { }
    private void m3() { }
}

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

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 e non produce output, 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 “Hello World” 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 “Hello World!”.*

Java Default Constructors

Cosa stampa questo programma?

E cosa stampa se viene eliminata la definizione del costruttore nella classe Padre?

```
class Padre {  
    Padre() { System.out.println("Padre!"); }  
}  
  
class Figlio extends Padre {  
    Figlio() { System.out.println("Figlio!"); }  
}  
  
class Example {  
    public static void main(String[] args){  
        Figlio p = new Figlio();  
    }  
}
```

Java Default Constructors

Risposta:

> Padre!

> Figlio!

*Se si togliesse il costruttore del Padre stamperebbe solo
“Figlio!”*

Static vs Dynamic Types (2)

Quali sono le istruzioni scorrette nel metodo main?

Una volta eliminate tali istruzioni, cosa stampa il programma?

Qual è il tipo statico e dinamico di ciascuna delle tre variabili al termine dell'esecuzione del main?

Static vs Dynamic Types (2)

```
public class Person {  
    public void greet() { System.out.println("Salve");}  
}  
public class EasyPerson extends Person {  
    public void greet() { System.out.println("Ciao"); }  
}  
public class FormalPerson extends Person {  
    public void greet() { System.out.println("Buongiorno"); }  
}  
public class VeryFormalPerson extends FormalPerson {  
    public void greet() { System.out.println("Salute"); }  
}
```


Static vs Dynamic Types (2)

```
class Example {  
    public static void main(String[] args) {  
        Person p = new Person();  
        EasyPerson ep = new EasyPerson();  
        FormalPerson fp = new FormalPerson();  
        VeryFormalPerson vfp=new VeryFormalPerson();  
        p.greet();                //1  
        ep = p;                   //2  
        p = ep;                   //3  
        p.greet();                //4  
        ep = fp;                  //5  
        ep.greet();               //6  
        fp.greet();               //7  
        p = new FormalPerson();   //8  
        p.greet();                //9  
        fp = p;                   //10  
        vfp = (VeryFormalPerson) fp; //11  
        vfp.greet();              //12  
    }  
}
```

Static vs Dynamic Types (2)

Risposta:

2, 5 e 10 sono **scorrette**: assegnamento di un sovra-tipo a un sotto-tipo o di un tipo non compatibile.

11 crea un **errore a runtime**: casting di un sovra-tipo verso una sottoclasse.

```
class Example {  
    public static void main(String[] args) {  
        Person p = new Person();  
        EasyPerson ep = new EasyPerson();  
        FormalPerson fp = new FormalPerson();  
        VeryFormalPerson fvp=new VeryFormalPerson();  
        p.greet(); //1  
        p = ep; //3  
        p.greet(); //4  
        ep.greet(); //6  
        fp.greet(); //7  
        p = new FormalPerson(); //8  
        p.greet(); //9  
        vfp = (VeryFormalPerson) fp; //11  
        vfp.greet(); //12  
    }  
}
```

Static vs Dynamic Types (2)

Risposta:

- > *Salve*
- > *Ciao*
- > *Ciao*
- > *Buongiorno*
- > *Buongiorno*

A questo punto l'esecuzione dell'istruzione 11 solleva un'eccezione, dal momento che il tipo dinamico di fp non e' VeryFormalPerson, e il programma termina.

Hierarchical Polygons

Definire una gerarchia di poligoni e sfruttare il poliformismo

Specifiche:

- *Polygon* è una classe astratta che definisce il metodo astratto *getPerimeter()*
- *Polygon* implementa una funzione *printPerimeters()* che stampi il perimetro di un array di poligoni
- Implementare le sotto-classi di *Polygon* *Square*, *Rectangle* e *Triangle*, ognuna con la propria implementazione di *getPerimeter()*

Secure String

Definire classi che permettano la stampa condizionata di una stringa

Specifiche:

- *SecureString* è una classe astratta che incapsula una stringa e che offre un metodo *securePrint(Object o)* che stampa la stringa dopo un controllo di sicurezza.
- Il controllo di sicurezza deve essere personalizzabile dalle sottoclassi
- Implementare una sottoclasse *CapabilitySecureString* di *SecureString* che offre un metodo *getCapability()* che ritorna un oggetto. Una volta chiamato, la stampa è possibile solo se l'oggetto passato a *securePrint* è quello ritornato da *getCapability()*

Runtime Type Checking

Cosa stampa questo programma?

```
class Father { }
class Son extends Father { }
class Test {
    public static void main(String[] s) {
        Father f = new Son();
        Father f2 = new Father();
        if (f instanceof Father)
            System.out.println("True");
        else
            System.out.println("False");
        if (f.getClass() == f2.getClass())
            System.out.println("True");
        else
            System.out.println("False");
    }
}
```

Runtime Type Checking

Risposta:

> *True*

> *False*

instanceof restituisce true se c'è compatibilità di assegnamento.

getClass() ritorna una reference alla (unica) istanza di tipo *Class* della classe dell'oggetto su cui è chiamato

Cron

Si progetti un package che offra un "demone temporale" simile a cron di Unix

Specifiche:

- L'utente del package deve poter creare un demone, registrare presso di lui una serie di coppie *<orario, azione da compiere>*
- Il demone temporale, una volta avviato, deve eseguire le azioni registrate all'orario prestabilito.
- Si supponga che non si possano registrare più di 10 azioni, che ogni azione debba venir eseguita una volta soltanto e che una volta eseguite tutte le azioni cron termini la sua esecuzione.
- Si può interpretare l'orario di esecuzione come "orario indicativo": viene garantito che l'azione viene eseguita **dopo** l'orario specificato