UNIVERSITÀ DEGLI STUDI DI BERGAMO
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

8. Domande di teoria

Vari esercizi simili a quelli che potrebbero capitare nella parte teorica da 45 minuti

Tutorato di Programmazione Avanzata

RELATORE
Imberti Federico

SEDE
Dalmine, BG

Binding Dinamico in Java

- Detto anche “Late binding”
  - Il metodo da eseguire viene stabilito a compile-time invece che a runtime

- Risoluzione esercizi in 3 fasi:
  1. Scrivere il **diagramma delle classi** della traccia (facoltativo ma altamente consigliato per non confondersi);

  2. Dai tipi assegnati a **compile-time** stabilire la segnatura del metodo da eseguire rispetto al reference type dell’oggetto;

  3. Dati tipi assegnati a **runtime** cercare la segnatura che corrisponde a quella stabilita al punto 2 rispetto alla gerarchia delle classi.

Binding Dinamico in Java

Quali sono i tipi a runtime e a compile-time?

- Il tipo a compile-time viene stabilito dal reference type

```
Persona p1;          //p1 è di tipo Persona
```

- Il tipo a runtime viene stabilito rispetto al tipo dell'oggetto che viene istanziato

```
p1 = new Studente() //p1 diventa di tipo Studente
```

Binding Dinamico in Java Esercizio 1/2

```
1 class Persona{}
2 class Studente extends Persona{}
3
4 class Scuola{
5   void iscrivi(Persona l) {
6     System.out.println("S");
7   }
8 }
9
10class Liceo extends Scuola{
11  void iscrivi(Persona l) {
12    System.out.println("L");
13  }
14}
15
16class University extends Scuola{
17  void iscrivi(Studente l) {
18    System.out.println("U");
19  }
20}
21
22...
23Persona p = new Studente();
24Studente s = new Studente();
25Scuola ss = new Scuola();
26Scuola sl = new Liceo();
27Scuola su = new University();
```

Cosa stampano le seguenti istruzioni?

A. ss.iscrivi(p);
B. sl.iscrivi(p);
C. su.iscrivi(p);
D. ss.iscrivi(s);
E. sl.iscrivi(s);
F. su.iscrivi(s);

Soluzioni

Binding Dinamico in Java Esercizio 2/2

```
1 class Elaboratore {
2   void setCPU(int l) {
3     System.out.println("E");
4   }
5 }
6
7 class Phone extends Elaboratore {
8   void setCPU(int l) {
9     System.out.println("P");
10  }
11}
12
13class Computer extends Elaboratore {
14  void setCPU(short l) {
15    System.out.println("C");
16  }
17}
18
19...
20Object oe = new Elaboratore ();
21Elaboratore ee = new Elaboratore ();
22Elaboratore ep = new Phone ();
23Elaboratore ec = new Computer ();
24short myfreq = 30;
```

Quale è l' input prodotto dalle sequenti istruzioni (errore se pensi ci sia un errore)?

oe.setCPU(myfreq) errore *(1 Punto)*
ee.setCPU(myfreq) E *(1 Punto)*
ep.setCPU(myfreq) P *(1 Punto)*
ec.setCPU(myfreq) E *(1 Punto)*

Activation Record

- Non è necessario restituire esattamente quello che indica il debugger
  - Ovvero non serve usare il computer per eseguirlo: potete inventare gli indirizzi

- Ricordate che state descrivendo lo Stack (LiFo)
  - Gli indirizzi scalano indietro solitamente di 4 posizioni per rappresentare una word (ogni cella ha 32 bit, cioè 4 byte)

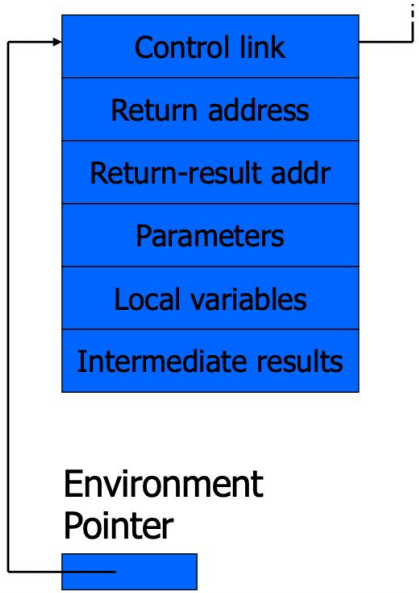
0x0…0FFFC -> 0x0…0FFF**8** -> 0x0…0FFF**4** -> 0x0…0FFF**0** -> 0x0…0FF**EC** -> 0x0…0FFE**8** -> …

- Allocate sempre spazio anche per i **valori intermedi** (eg. valutazione di un parametro) e per il **risultato** di una funzione (eg. funzioni ricorsive)

- Il **preambolo** di una funzione vuole che sia salvato il Link Register (**LR**) e il Frame Pointer (**FP**)

Activation Record

- Potete basarvi su questo schema, purché ogni compilatore abbia le sue particolarità e perciò non lo segua alla lettera



- Return address (Link Register)
  - Location of code to execute on function return
- Return-result address
  - Address in activation record of calling block to receive return address
- Parameters
  - Locations to contain data from calling block

Costruttori e distruttori in C++

```
1 struct A {
2              A() { cout << "A+"; }
3     virtual ~A() { cout << "A-"; }
4 };
5
6 struct B {
7      B() { cout << "B+"; }
8     ~B() { cout << "B-"; }
9 };
10
11struct C {
12     C() { cout << "C+"; }
13    ~C() { cout << "C-"; }
14};
15
16struct X : A, B, protected C {
17     X() { cout << "X+"; }
18    ~X() { cout << "X-"; }
19};
```

Scrivi l'output delle seguenti istruzioni (ERR se pensi che ci sia un errore):

A* a = new A; delete a;A+A- ***(1 Punto)***

A* a = new X; delete a;A+B+C+X+X-C-B-A- ***(1 Punto)***

B* b = new B; delete b;B+B- ***(1 Punto)***

B* b = new X; delete b;A+B+C+X+B- ***(1 Punto)***

C* c = new C; delete c;C+C- ***(1 Punto)***

C* c = new X; delete c;ERR ***(1 Punto)***

X* x = new X; delete x;A+B+C+X+X-C-B-A- ***(1 Punto)***

Ridefinizione di metodi in Java

```
1 class Veicolo{
2    public Veicolo get(){return null;}
3 }
4 class Auto extends Veicolo {
5    public Veicolo get(){return null;}
6 }
7 class Bicicletta extends Veicolo {
8    private Veicolo get(){return null;}
9 }
10class Autobus extends Veicolo {
11   public Autobus get(){return null;}
12}
```

Quali di questi metodi sono redefiniti in modo sbagliato (errore in compilazione)?

- ☐ **il metodo get di Auto** ***(Selezionato = -1 Punto, Non selezionato = 1 Punto)***
- ☐ **il metodo get di Autobus** ***(Selezionato = -1 Punto, Non selezionato = 1 Punto)***
- ☑ **il metodo get di Bicicletta** ***(Selezionato = 1 Punto, Non selezionato = -1 Punto)***

Passaggio di array in C

```
1 void f(int a[], int n) {
2         printf("%d\n", n);
3         printf("%d\n", sizeof(a));
4         a = a + 1;
5         *a = *a + 1;
6 }
7
8 int main(void) {
9         int p[] = { 10, 20, 30 };
10        f(p, sizeof(p));
11        printf("%d\n", *p );
12        return 0;
13}
```

Qual è l'output prodotto dalle seguenti istruzioni (in ordine di esecuzione)? Se pensi contenga un errore, scrivi errore

Assumi che un puntatore vale 4 byte come anche un intero (32 bit).

in f:

printf("%d\n", n);12 ***(1 Punto)***
printf("%d\n", sizeof(a));4 ***(4 Punti)***

in main:

printf("%d\n", *p);10 ***(2 Punti)***

Virtual functions ed ereditarietà in C++

```
1 class X {
2 private:
3     void pri() { cout << "X" << endl; }
4 public:
5     virtual void pub() { cout << "X" << endl; }
6 };
7
8 class Y : private X {
9 public:
10    void pri() { cout << "Y" << endl; }
11
12};
13
14class Z : private X {
15public:
16    virtual void pub() { cout << "Z" << endl; }
17};
18
19class V : public X {
20public:
21    void pri() { cout << "V" << endl; }
22};
23
24class W : public X {
25public:
26    virtual void pub() { cout << "W" << endl; }
27};
```

X x; x.pri();ERR *(1 Punto)*
Y y; y.pri();Y *(1 Punto)*
Z z; z.pri();ERR *(1 Punto)*
V v; v.pri();V *(1 Punto)*
W w; w.pri();ERR *(1 Punto)*
x = y; x.pub();ERR *(1 Punto)*
x = z; x.pub();ERR *(1 Punto)*
x = v; x.pub();X *(1 Punto)*

x = w; x.pub();X *(1 Punto)*
X* p = &x; p->pub();X *(1 Punto)*
p = &y; p->pub();ERR *(1 Punto)*
p = &z; p->pub();ERR *(1 Punto)*
p = &v; p->pub();X *(1 Punto)*
p = &w; p->pub();W *(1 Punto)*

Overriding Vs Overloading in Java

```
1 class Value {}
2 class SmallValue extends Value {}
3
4 class Elaboratore {
5     Value getVal() {
6         return new Value();
7     }
8 }
9
10class Phone extends Elaboratore {
11    SmallValue getVal() {
12        return new SmallValue();
13    }
14}
```

Quali di queste affermazioni sono giuste?

- [x] **Phone fa overriding del metodo getVal di Elaboratore**
- [ ] **Phone contiene un errore: non può definire getVal in questo modo!**
- [x] **Phone è una sottoclasse di Elaboratore**
- [ ] **Phone fa overloading del metodo getVal di Elaboratore**

Esercizi della parte teorica con soluzioni

Esempi delle domande di teoria prese dai vecchi temi di Gargantini