DYNAMIC DISPATCH

Cosa significa?

- Il tipo di una variabile non determina in modo univoco il tipo dell'oggetto che la variabile riferisce
- Cerchiamo di capire quale e' il problema
- Class B extends class A
  // L'estensione riscrive il metodo m()
- Supponiamo di creare un oggetto di tipo A
  A a = new A()
  e di fare diverse operazioni su a.
- Invochiamo a.m(). Quale metodo effettivamento chiamiamo?

Esempio

```
public class DynamicBindingTest {

   public static void main(String args[]) {
      Vehicle vehicle = new Car(); // Il tipo statico e'  Vehicle ma
                                   // il tipo dinamico e' Car
      vehicle.start();     //Quale metodo viene invocato?
                           // Quello di Car o quello di Vehicle?
   }
}

class Vehicle {
   public void start() {
      System.out.println("Inside start method of Vehicle");
   }
}

class Car extends Vehicle {
   public void start() {
      System.out.println("Inside start method of Car");
   }
}
```

```
public class Test {

   public static void main(String args[]) {
      Vehicle vehicle = new Car();
      vehicle.start();     //Quale metodo viene invocato?
                           // Quello di Car o quello di Vehicle?
   }
}

class Vehicle {
   public void start() {
      System.out.println("Inside start method of Vehicle");
   }
}

class Car extends Vehicle {
   public void start() {
      System.out.println("Inside start method of Car");
   }
}
```

Output:
Inside start method of Car

```
public class Counter {
  private int x;
  public Counter () { x = 0; }
  public void incBy(int d) { x = x + d; }
  public int get() { return x; }
}

public class Decr extends Counter {
  private int y;
  public Decr (int initY) {y = initY; }
  public void dec() { incBy(-y);  }
}
```

```
//nel main troviamo
Decr d = new Decr(2);
d.dec();
int x = d.get();
```

Intuizione

- Invocazione **o.m()**
- A tempo di esecuzione viene utilizzato il tipo dinamico dell'oggetto **o** per determinare nella gerarchia delle classi quale e' il metodo piu' specifico da invocare

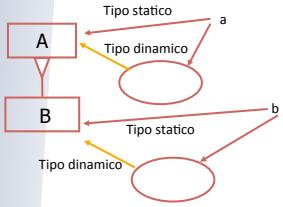
- B e' un *sottotipo* di A
- A e B includono la definizione del metodo m

```
A a = new A ();
B b = new B ();

a.m();
b.m();
a = b;
a.m()
```

Viene invocato il metodo della classe A
Viene Invocato il metodo della classe B
Viene invocato il metodo della classe B

Dynamic Dispatch

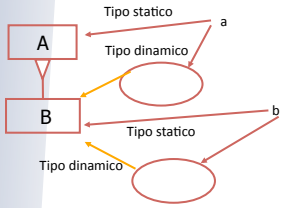
- Viene ricercato il metodo esaminando la gerarchia a partire dal tipo **dinamico** dell'oggetto




```
A a = new A ();
B b = new B ();

a.m();
b.m();
```


Dynamic Dispatch

- Viene ricercato il metodo lungo la gerarchia a partire dal tipo **dinamico** dell'oggetto




```
A a = new A ();
B b = new B ();

a.m();
b.m();
a = b;
```

Tipo statico di a e' A
Tipo dinamico di a e' B

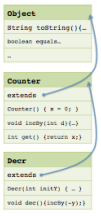
Static vs dynamic

- Compilalatore utilizza i tipi statici per determinare la correttezza delle invocazioni dei metodi
- La macchina virtuale utilizza il tipo dinamico per determinare l'effettivo metodo da invocare

La tabella dei metodi

- Per comprendere gli esempi precedenti di deve estendere la ASM di Java con una nuova componente: la *tabella dei metodi* (a volte chiamata anche *tabella della classe*)
- La tabella contiene il codice dei metodi definiti nella classe, e tutte le componenti statiche definite nella classe stessa.
- La tabella contiene inoltre un puntatore alla classe padre.
- L'insieme delle tabelle e' pertanto un albero (perche?)

Esempio




```
public class Counter {
  private int x;
  public Counter () { x = 0; }
  public void incBy(int d) { x = x + d; }
  public int get() { return x; }
}

public class Decr extends Counter {
  private int y;
  public Decr (int initY) {y = initY; }
  public void dec() { incBy(-y);  }
}
```

Tabella dei metodi

- Le tabelle dei metodi sono allocate sullo heap (memoria dinamica)
- L'invocazione del costruttore determina l'allocazione sullo heap della tabella dei metodi associata alla classe dell'oggetto creato (se non e' gia' presente)
- Ogni oggetto sullo heap contiene un puntore alla tabella dei metodi del suo tipo **dinamico**

Dispatch

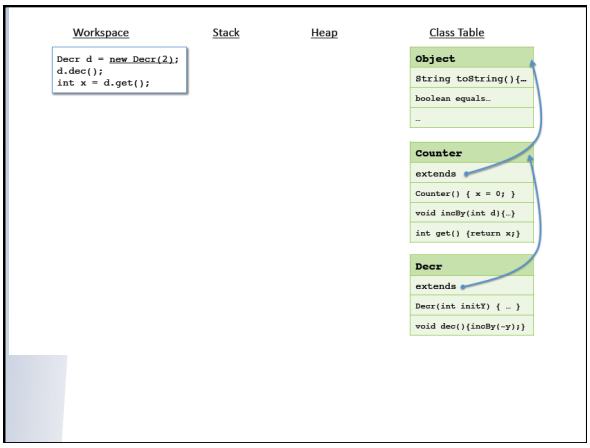
- L'invocazione del metodo

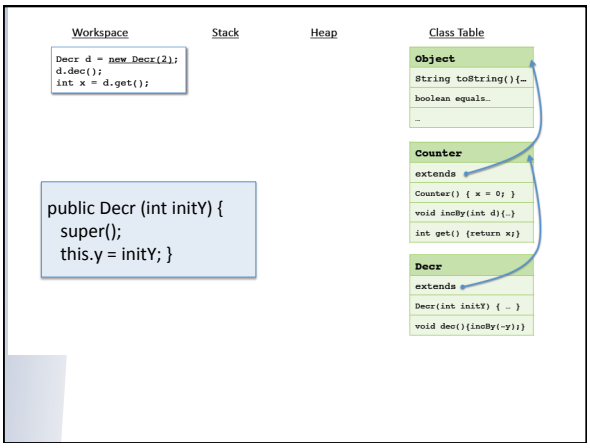
  o.m()

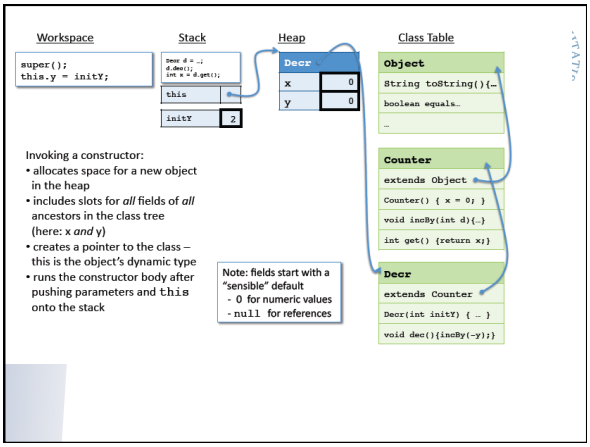
  utilizza il puntatore alla tabella dei metodi per effettuare l'operazione di dispatch
  - Ricerca nella gerachia a partire dalla tabella dei metodi associata al tipo dinamico dell'oggetto o.
  - Da notare l'utilizzo di **this** per determinare l'oggetto che invoca il metodo

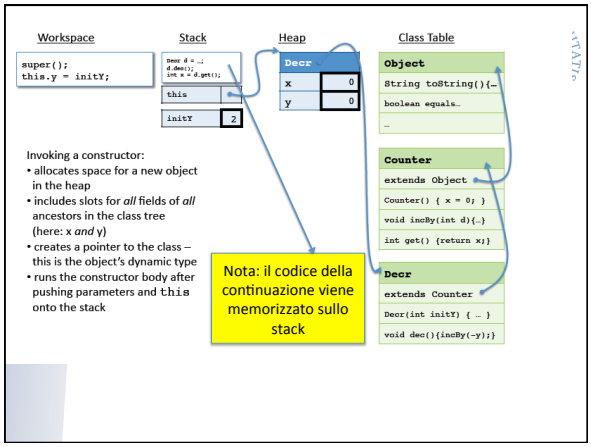
```
public class Counter extends Object {
  private int x;
  public Counter () {
   super();
   this.x = 0; }
  public void incBy(int d) {
   this.x = this.x + d;}
  public int get() {return this.x; }
}

public class Decr extends Counter {
  private int y;
  public Decr (int initY) {
   super();
   this.y = initY; }
  public void dec() {
   this.incBy(-this.y);}
}


//  nel main:
Decr d = new Decr(2); d.dec();
int x = d.get();
```

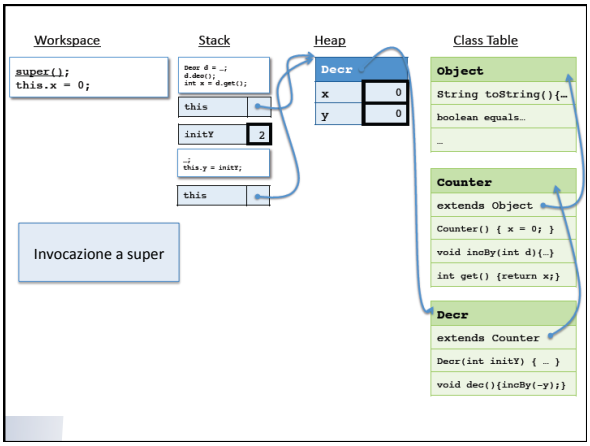
ANIMAZIONE DELL'ESECUZIONE

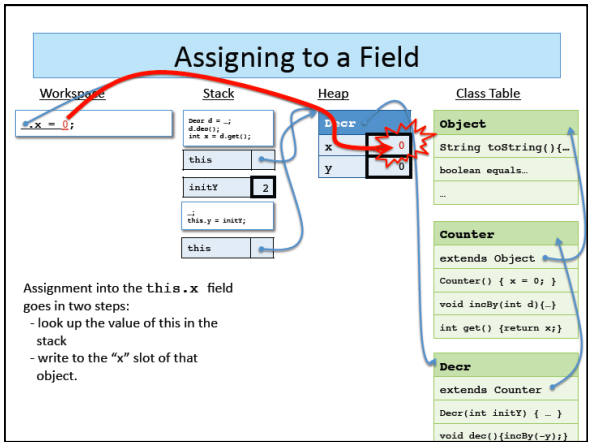


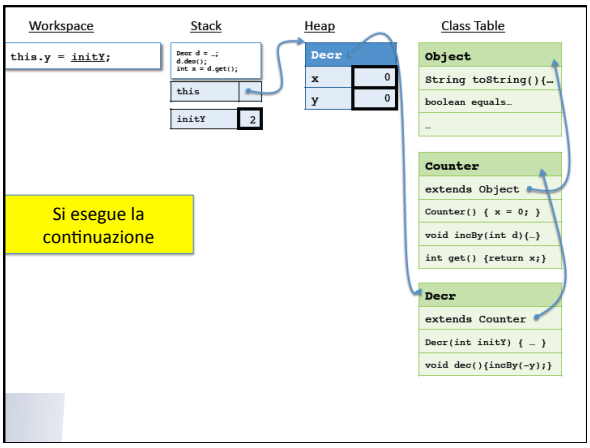


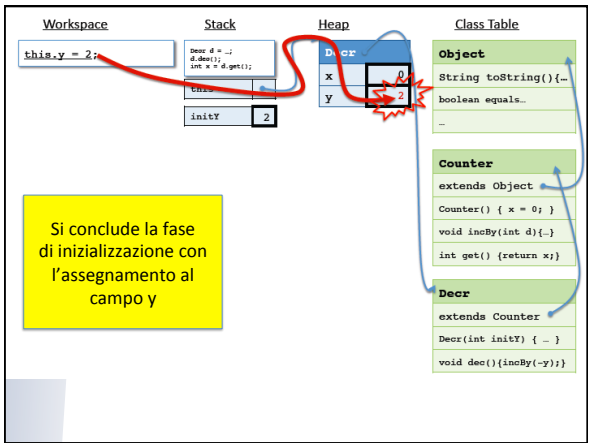

Workspace
Stack
Heap
Class Table
super();
this.y = initY;
this
initY
2
Decr
x
0
y
0
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invoking a constructor:
• allocates space for a new object in the heap
• includes slots for all fields of all ancestors in the class tree (here: x and y)
• creates a pointer to the class – this is the object's dynamic type
• runs the constructor body after pushing parameters and this onto the stack
Note: fields start with a "sensible" default
- 0 for numeric values
- null for references

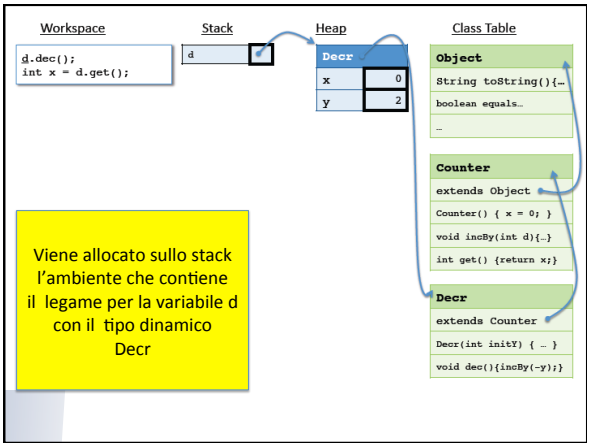

Workspace
Stack
Heap
Class Table
super();
this.y = initY;
this
initY
2
Decr
x
0
y
0
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invoking a constructor:
• allocates space for a new object in the heap
• includes slots for all fields of all ancestors in the class tree (here: x and y)
• creates a pointer to the class – this is the object's dynamic type
• runs the constructor body after pushing parameters and this onto the stack
Nota: il codice della continuazione viene memorizzato sullo stack

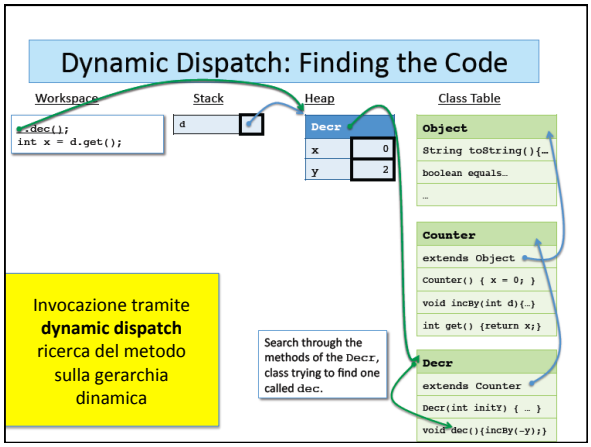

Workspace
Stack
Heap
Class Table
super();
this.x = 0;
this
initY
2
this
Decr
x
0
y
0
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invocazione a super

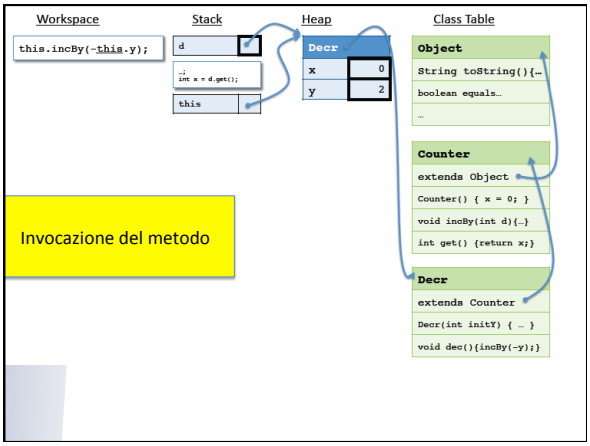

Assigning to a Field
Workspace
Stack
Heap
Class Table
this.x = 0;
this
initY
2
this
Decr
x
0
y
0
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Assignment into the this.x field goes in two steps:
- look up the value of this in the stack
- write to the "x" slot of that object.

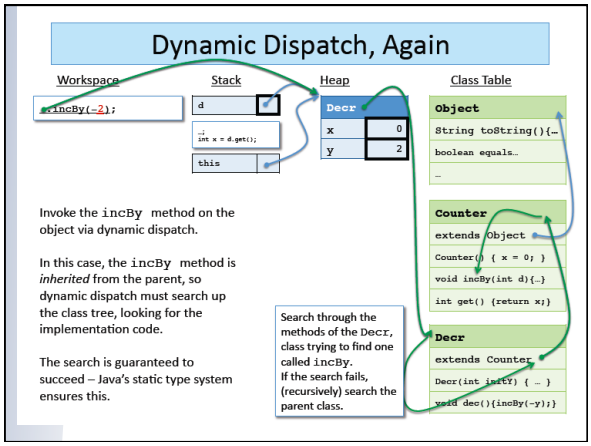

Workspace
Stack
Heap
Class Table
this.y = initY;
this
initY
2
Decr
x
0
y
0
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Si esegue la continuazione

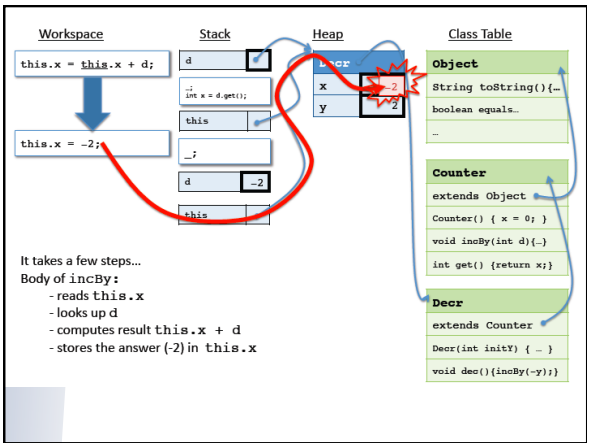

Workspace
Stack
Heap
Class Table
this.y = 2;
this
initY
2
Decr
x
0
y
2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Si conclude la fase di inizializzazione con l'assegnamento al campo y

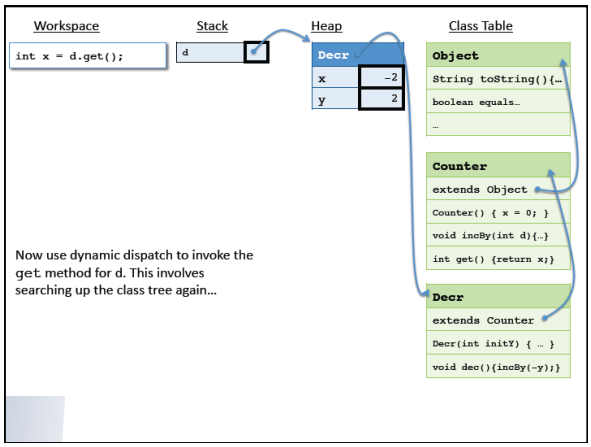

Workspace
Stack
Heap
Class Table
d.dec();
int x = d.get();
d
Decr
x 0
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Viene allocato sullo stack l'ambiente che contiene il legame per la variabile d con il tipo dinamico Decr


Dynamic Dispatch: Finding the Code
Workspace
Stack
Heap
Class Table
.dec();
int x = d.get();
d
Decr
x 0
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invocazione tramite dynamic dispatch ricerca del metodo sulla gerarchia dinamica
Search through the methods of the Decr, class trying to find one called dec.


Workspace
Stack
Heap
Class Table
this.incBy(-this.y);
d
int x = d.get();
this
Decr
x 0
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invocazione del metodo


Dynamic Dispatch, Again
Workspace
Stack
Heap
Class Table
incBy(-2);
d
int x = d.get();
this
Decr
x 0
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Invoke the incBy method on the object via dynamic dispatch.
In this case, the incBy method is inherited from the parent, so dynamic dispatch must search up the class tree, looking for the implementation code.
The search is guaranteed to succeed – Java's static type system ensures this.
Search through the methods of the Decr, class trying to find one called incBy. If the search fails, (recursively) search the parent class.


Workspace
Stack
Heap
Class Table
this.x = this.x + d;
this.x = -2;
d
int x = d.get();
this
_;
d -2
this
Decr
x -2
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
It takes a few steps…
Body of incBy:
- reads this.x
- looks up d
- computes result this.x + d
- stores the answer (-2) in this.x


Workspace
Stack
Heap
Class Table
int x = d.get();
d
Decr
x -2
y 2
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
Now use dynamic dispatch to invoke the get method for d. This involves searching up the class tree again…