



AA 2014-2015

PROGRAMMAZIONE 2

6.Dynamic dispatch

Cosa significa?



- ✎ La dichiarazione di una variabile non determina in modo univoco il tipo dell'oggetto che la variabile riferisce
- ✎ Cerchiamo di capire quale è 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` che coinvolgono anche aliasing con oggetti di tipo B. Poi invochiamo `a.m()`. Quale metodo è effettivamente invocato?

Esempio



```
public class DynamicBindingTest {
    public static void main(String args[]) {
        Vehicle vehicle = new Car( ); // Il tipo statico è Vehicle
                                     // il tipo dinamico è 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

Intuizione



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



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

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

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

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

Viene invocato il metodo della classe A

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

Viene Invocato il metodo della classe B

```
a = b;
```

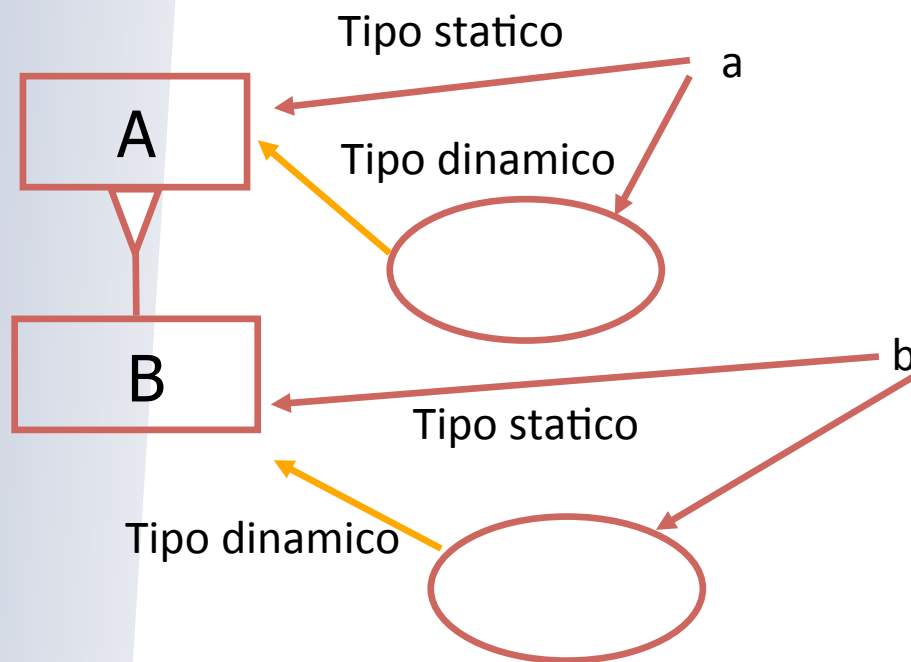
```
a.m( );
```

Viene invocato il metodo della classe B

Dynamic Dispatch



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



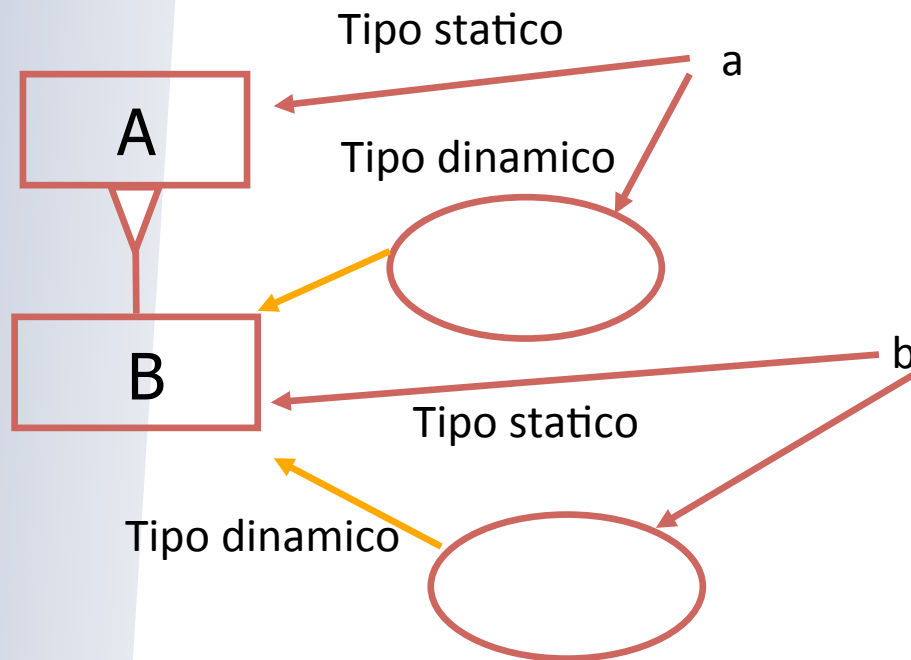
```
A a = new A( );  
B b = new B( );
```

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

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 è A
Tipo dinamico di a è B

Static vs dynamic



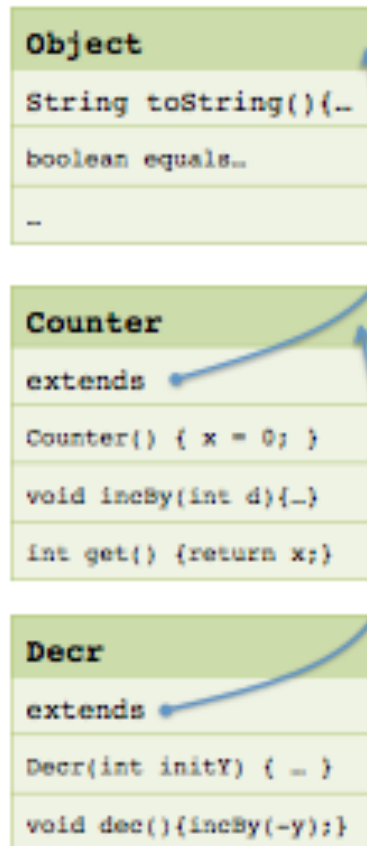
- ✎ Il compilatore usa i tipi statici per determinare la correttezza delle invocazioni dei metodi
- ✎ La macchina virtuale usa il tipo dinamico per determinare l'effettivo metodo da invocare

La tabella dei metodi



- ✎ Per comprendere gli esempi precedenti va estesa la ASM di Java con un'ulteriore componente: la tabella dei metodi (a volte chiamata 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 un puntatore alla classe padre
- ✎ L'insieme delle tabelle è pertanto un albero (perché?)

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) { super( ); y = initY; }
    public void dec( ) { incBy(-y); }
}
```

Tabella dei metodi



- ✉ Le tabelle dei metodi sono allocate sullo heap (memoria dinamica)
- ✉ L'invocazione del metodo costruttore determina l'allocazione sullo heap della tabella dei metodi associata alla classe dell'oggetto creato (se non è già 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 sulla gerarchia dell'oggetto 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

```
Decr d = new Decr(2);  
d.dec();  
int x = d.get();
```

Stack

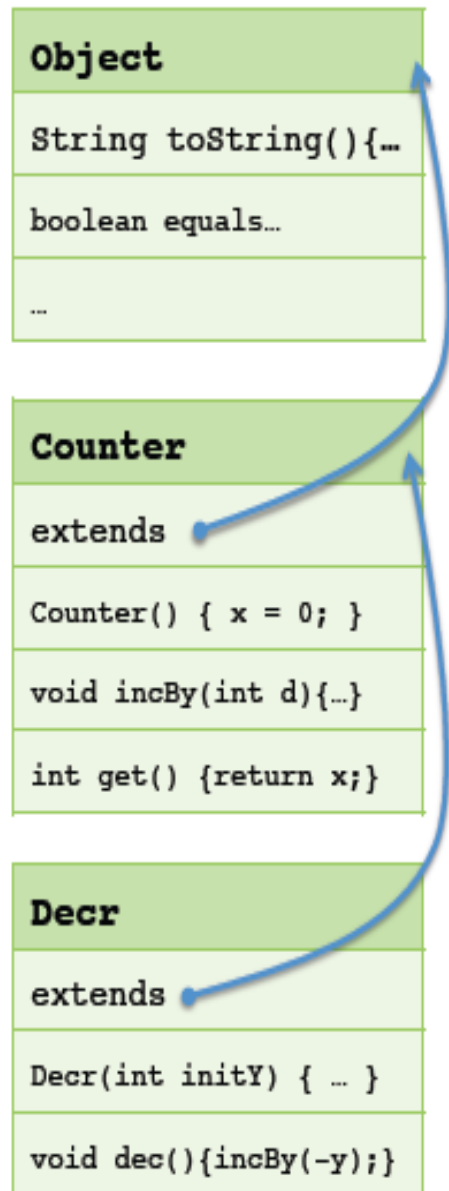
Heap

Class Table

Object
String toString(){...}
boolean equals...
...

Counter
extends
Counter() { x = 0; }
void incBy(int d){...}
int get() {return x;}

Decr
extends
Decr(int initY) { ... }
void dec(){incBy(-y);}



Workspace

```
Decr d = new Decr(2);  
d.dec();  
int x = d.get();
```

Stack

Heap

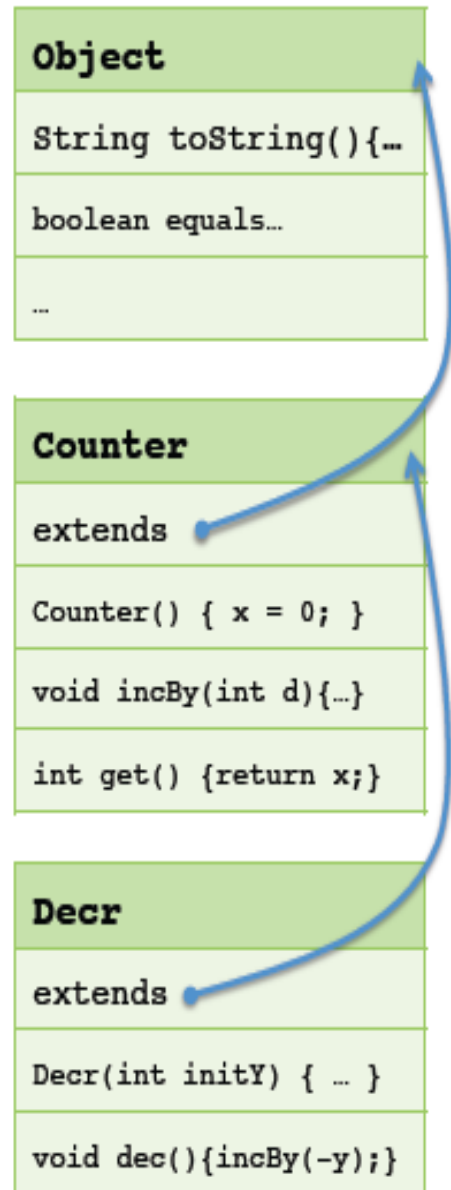
Class Table

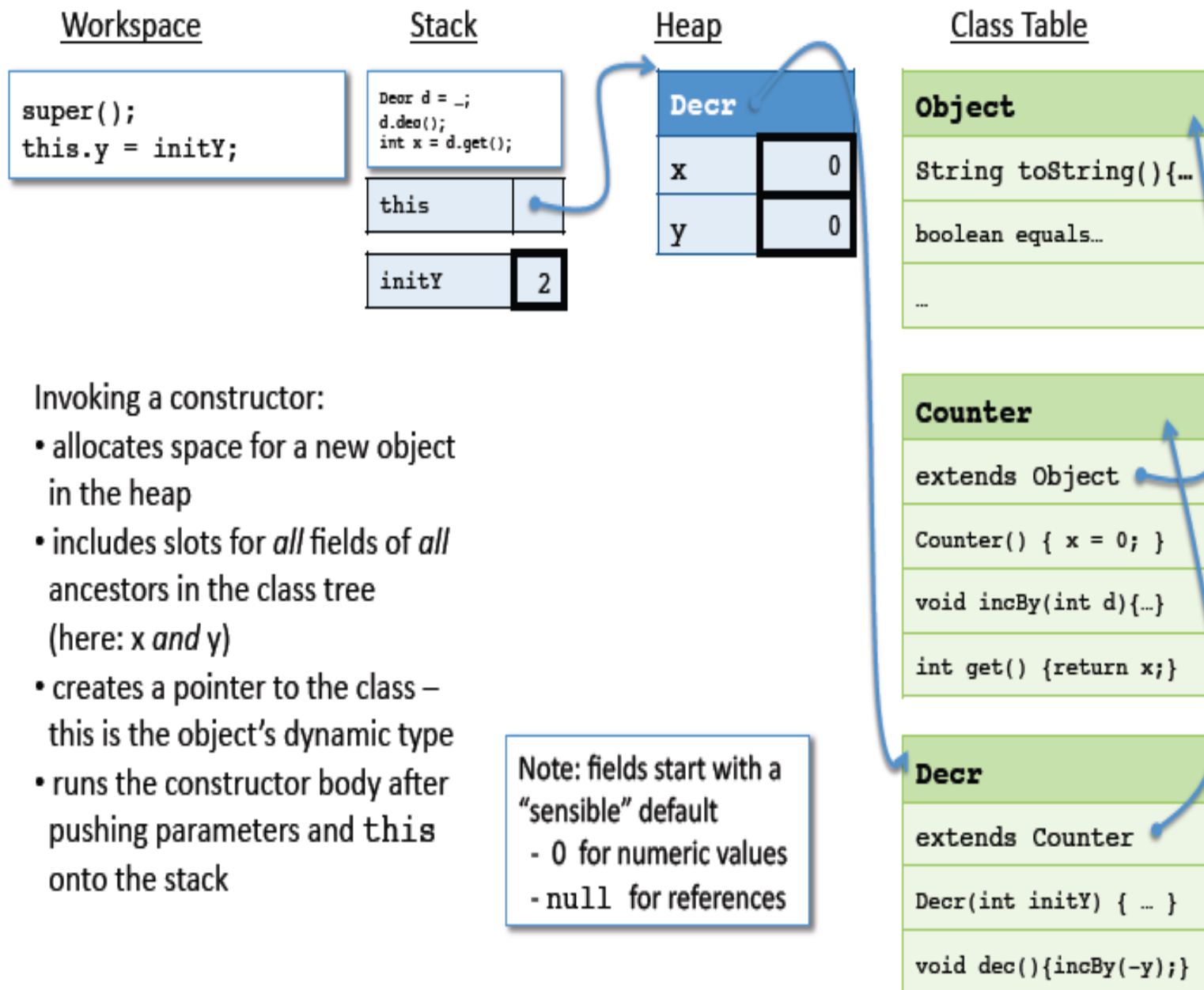
```
public Decr (int initY) {  
    super( );  
    this.y = initY;  
}
```

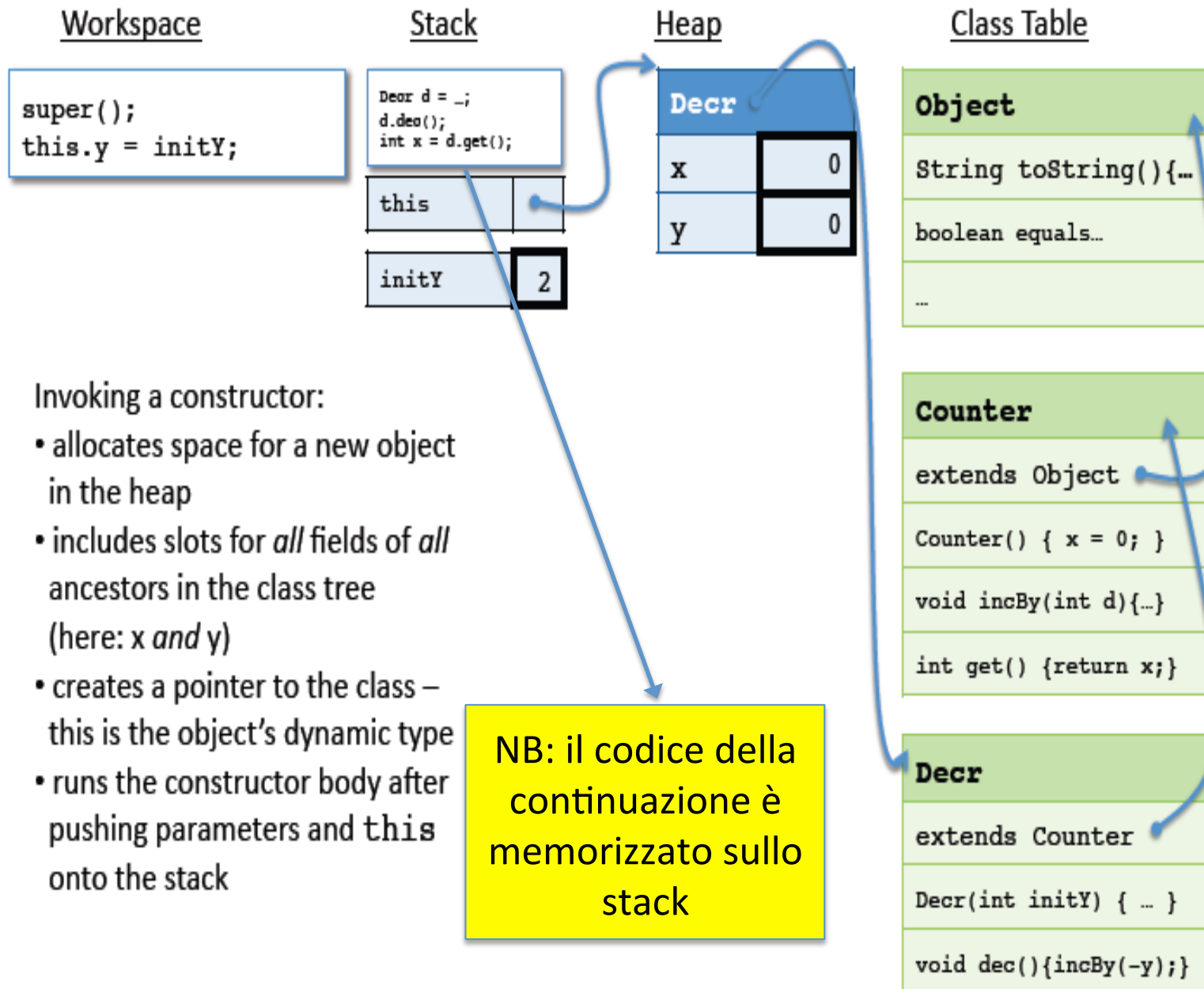
Object
String toString(){...}
boolean equals...
...

Counter
extends
Counter() { x = 0; }
void incBy(int d){...}
int get() {return x;}

Decr
extends
Decr(int initY) { ... }
void dec(){incBy(-y);}







Workspace

```
super();  
this.x = 0;
```

Stack

```
Decr d = _;  
d.dec();  
int x = d.get();
```

this	
------	--

initY	2
-------	---

```
_  
this.y = initY;
```

this	
------	--

Heap

Decr	
x	0
y	0

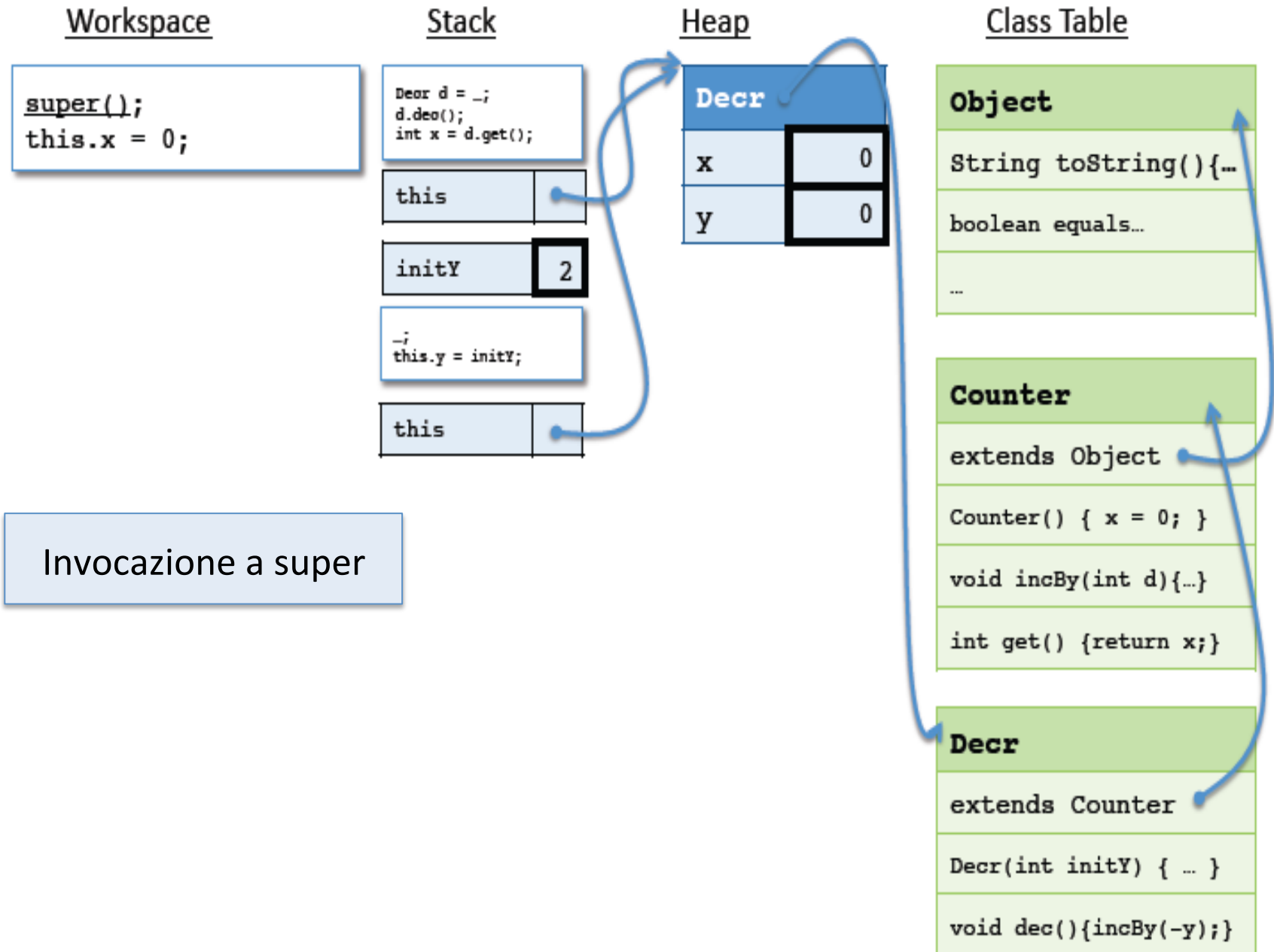
Class Table

Object
String toString(){...}
boolean equals...
...

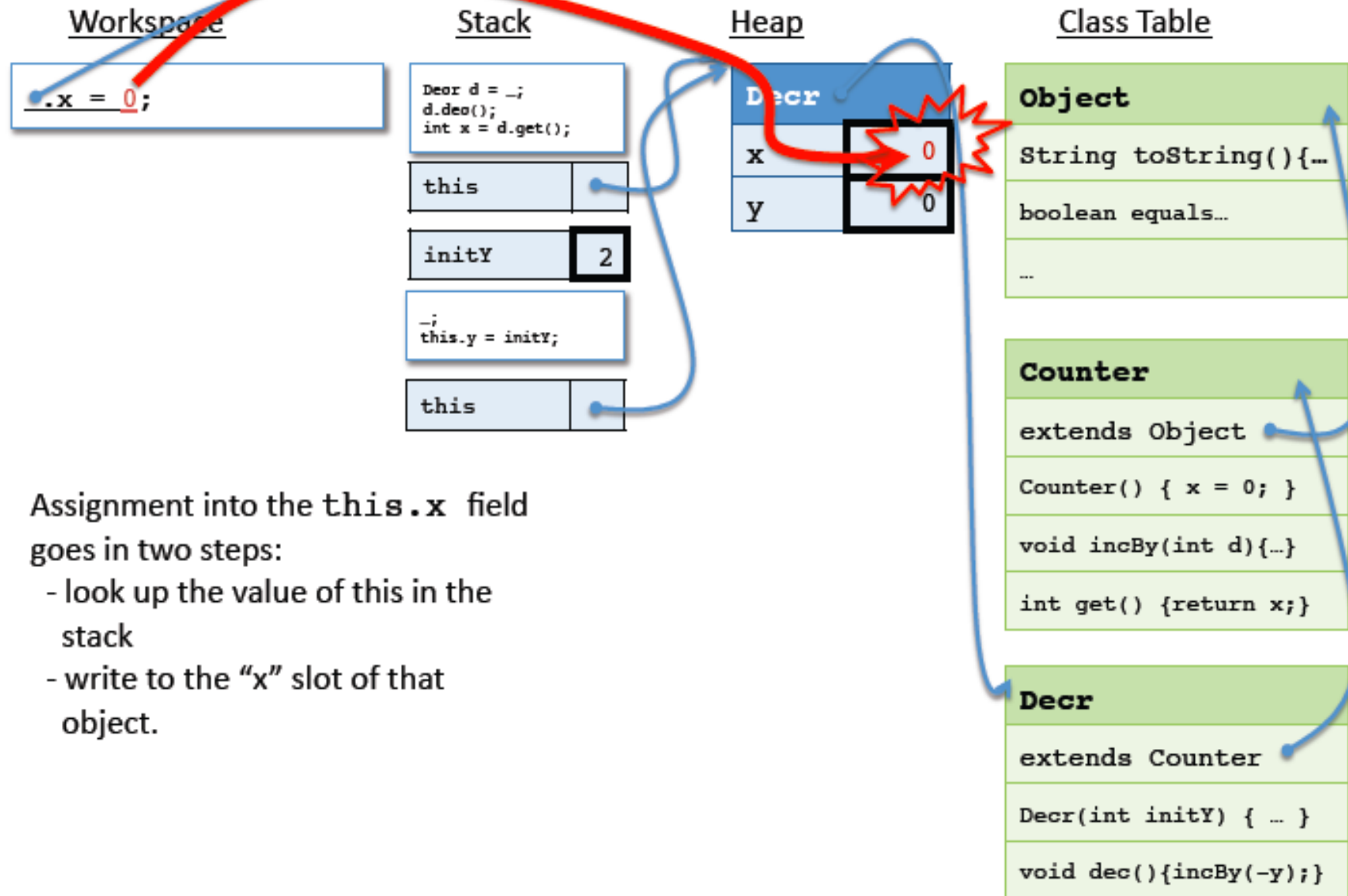
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){...}
int get() {return x;}

Decr
extends Counter
Decr(int initY) { ... }
void dec(){incBy(-y);}

Invocazione a super



Assigning to a Field



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

```
this.y = initY;
```

Stack

Decr d = _; d.dec(); int x = d.get();	
this	→
initY	2

Heap

Decr	
x	0
y	0

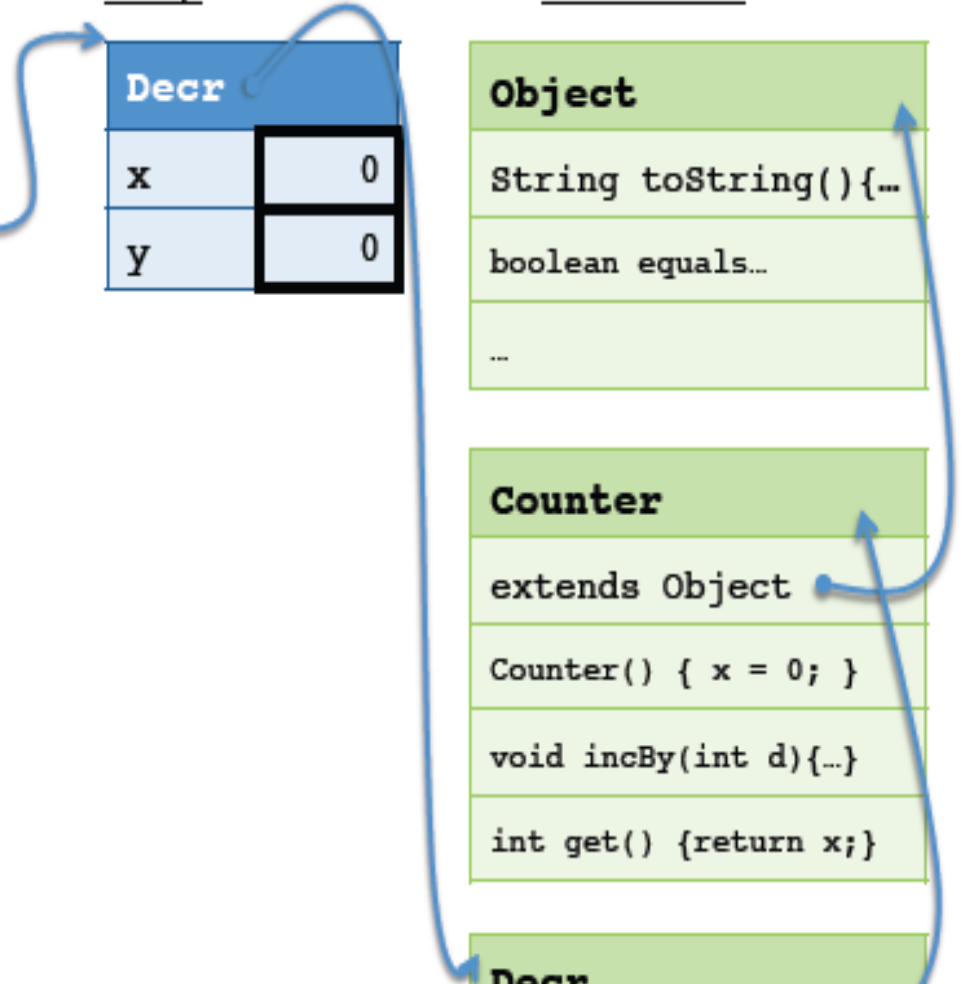
Class Table

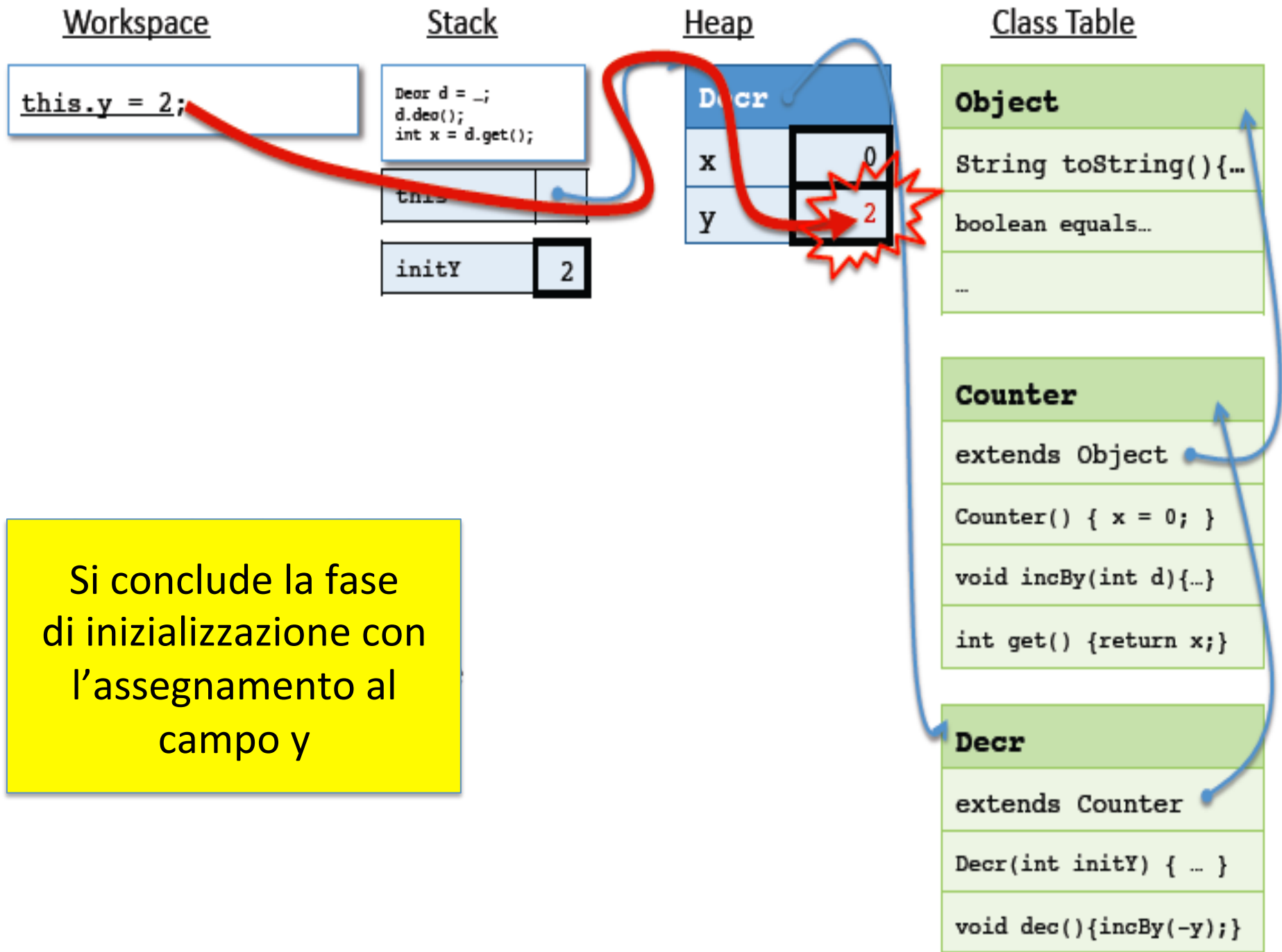
Object
String toString(){...}
boolean equals...
...

Counter
extends Object
Counter() { x = 0; }
void incBy(int d){...}
int get() {return x;}

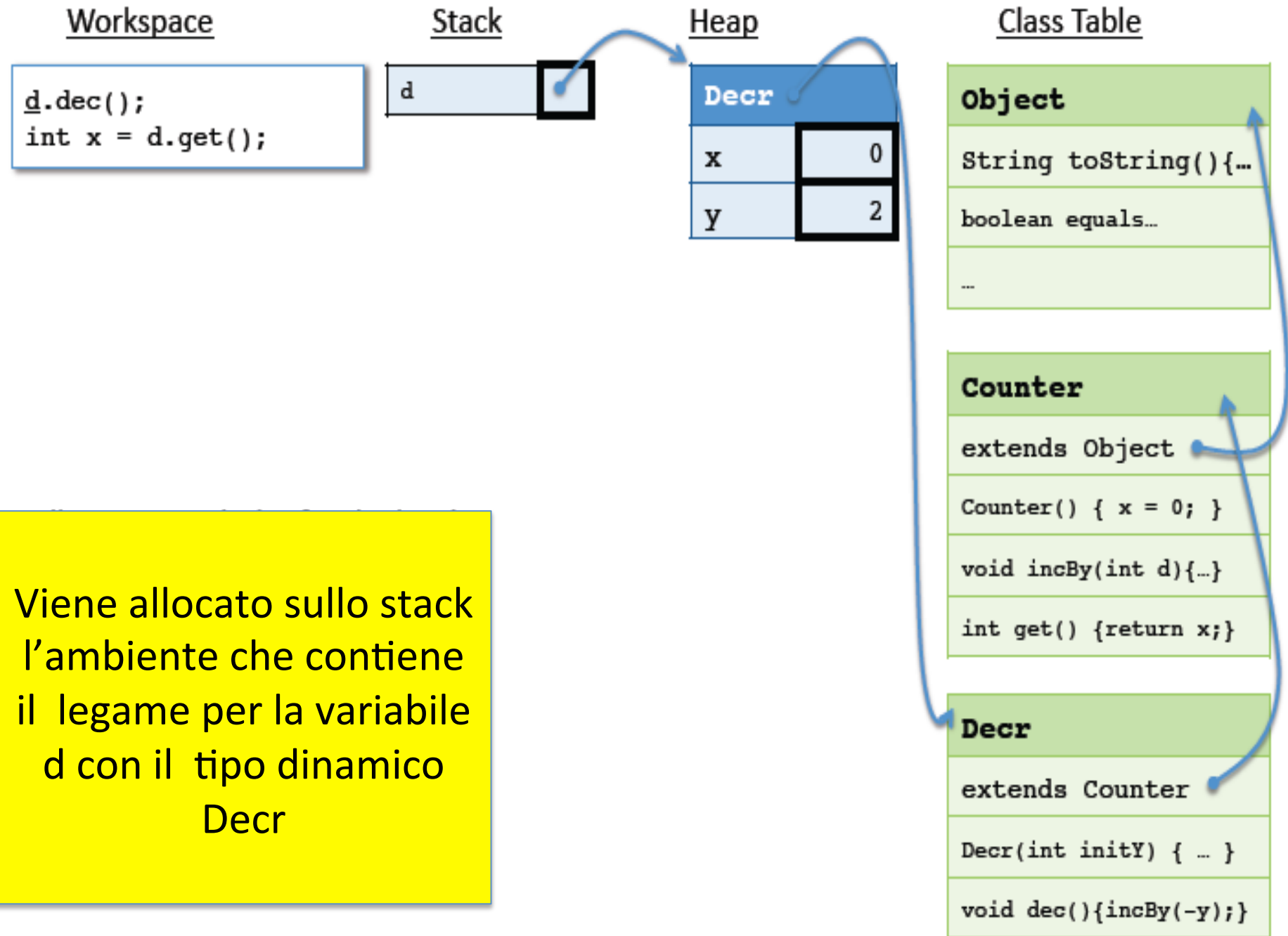
Decr
extends Counter
Decr(int initY) { ... }
void dec(){incBy(-y);}

Si esegue la
continuazione



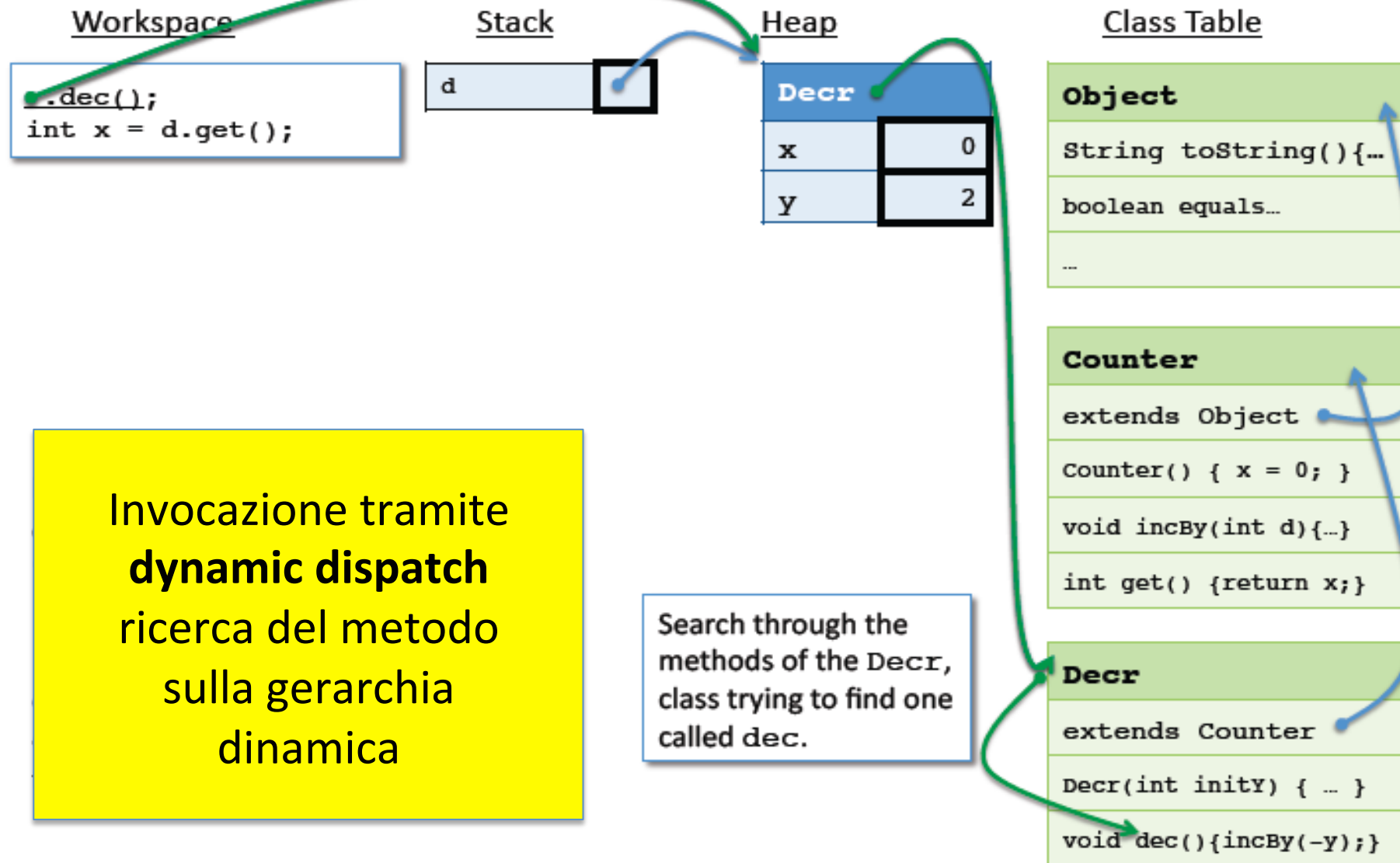


Si conclude la fase di inizializzazione con l'assegnamento al campo y



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

```
this.incBy(-this.y);
```

Stack

d	
-;	int x = d.get();
this	

Heap

Decr	
x	0
y	2

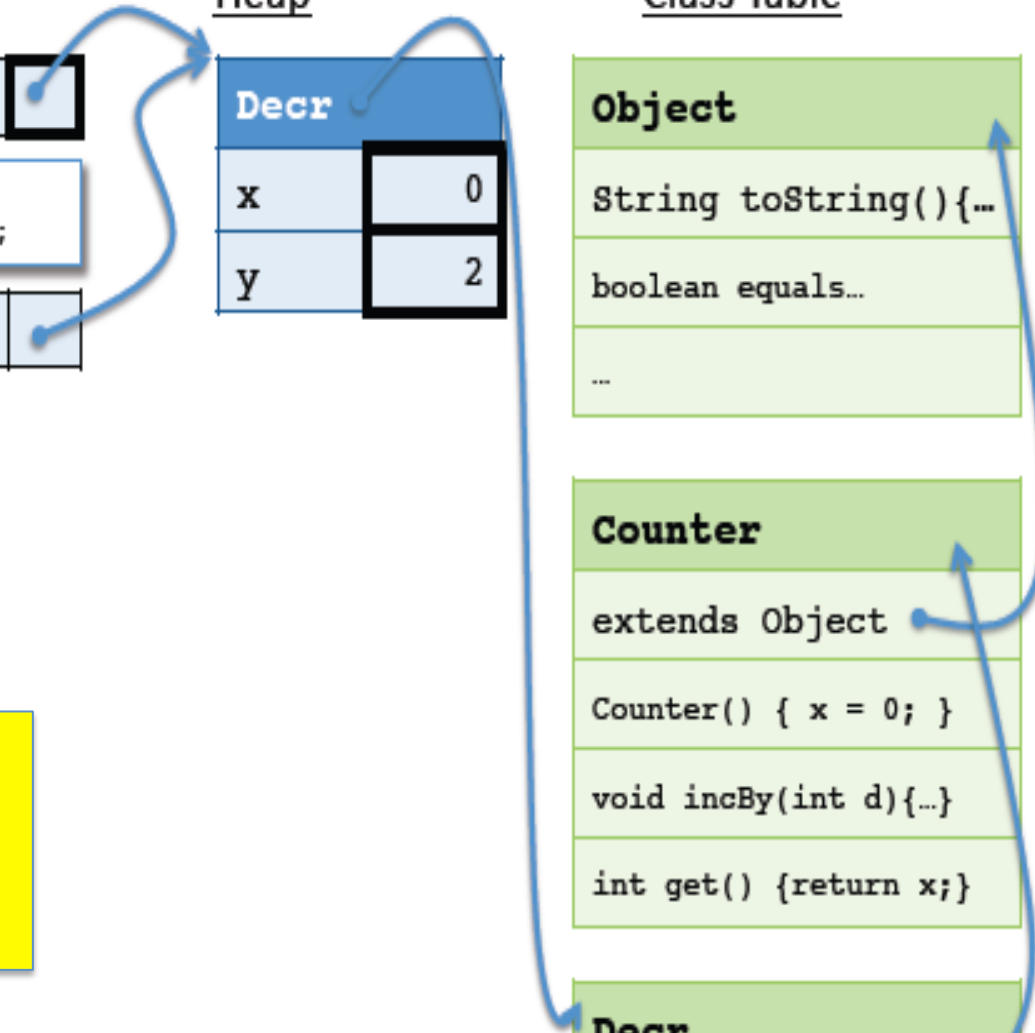
Class Table

Object
String toString(){...}
boolean equals...
...

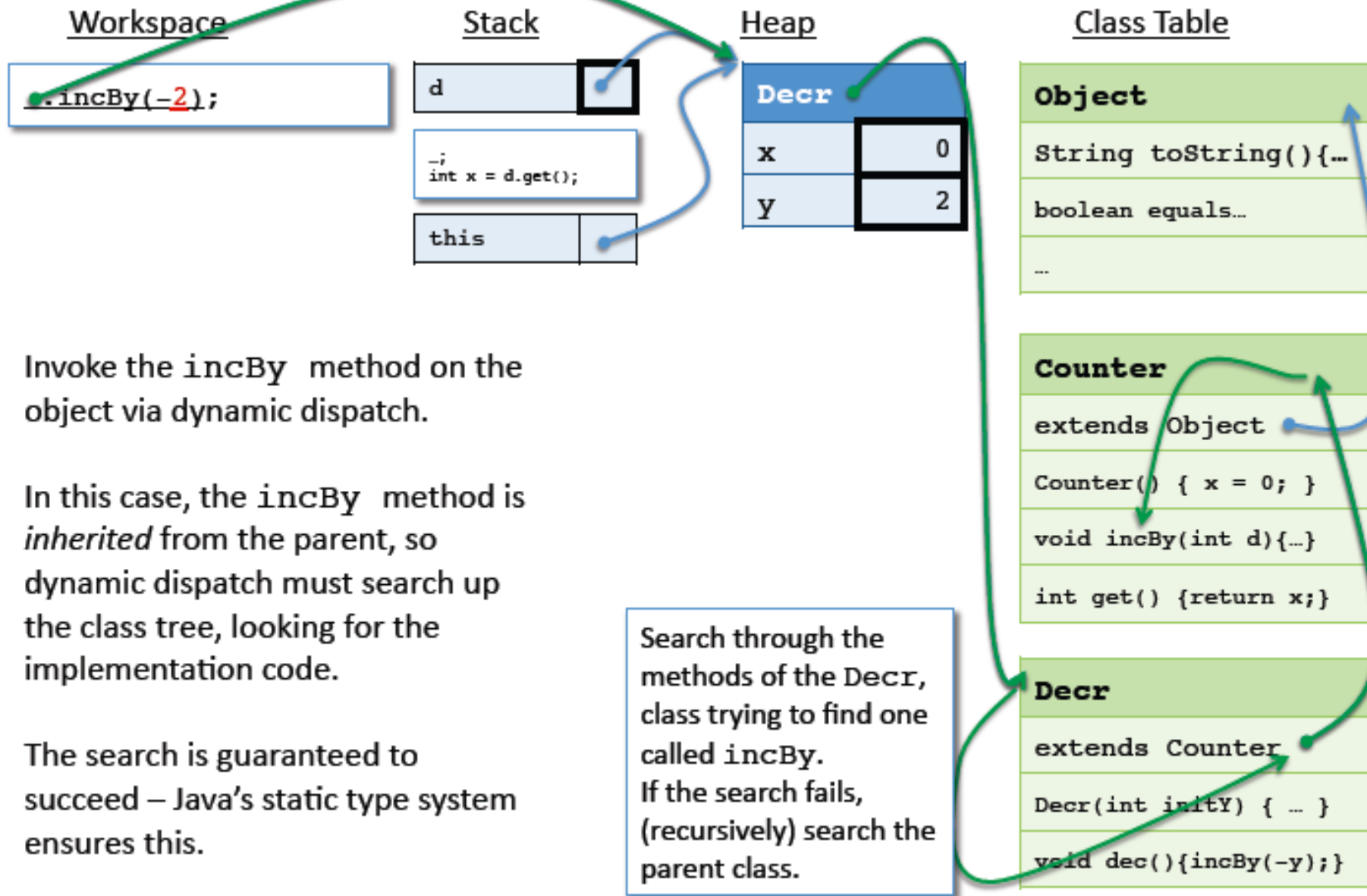
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){...}
int get() {return x;}

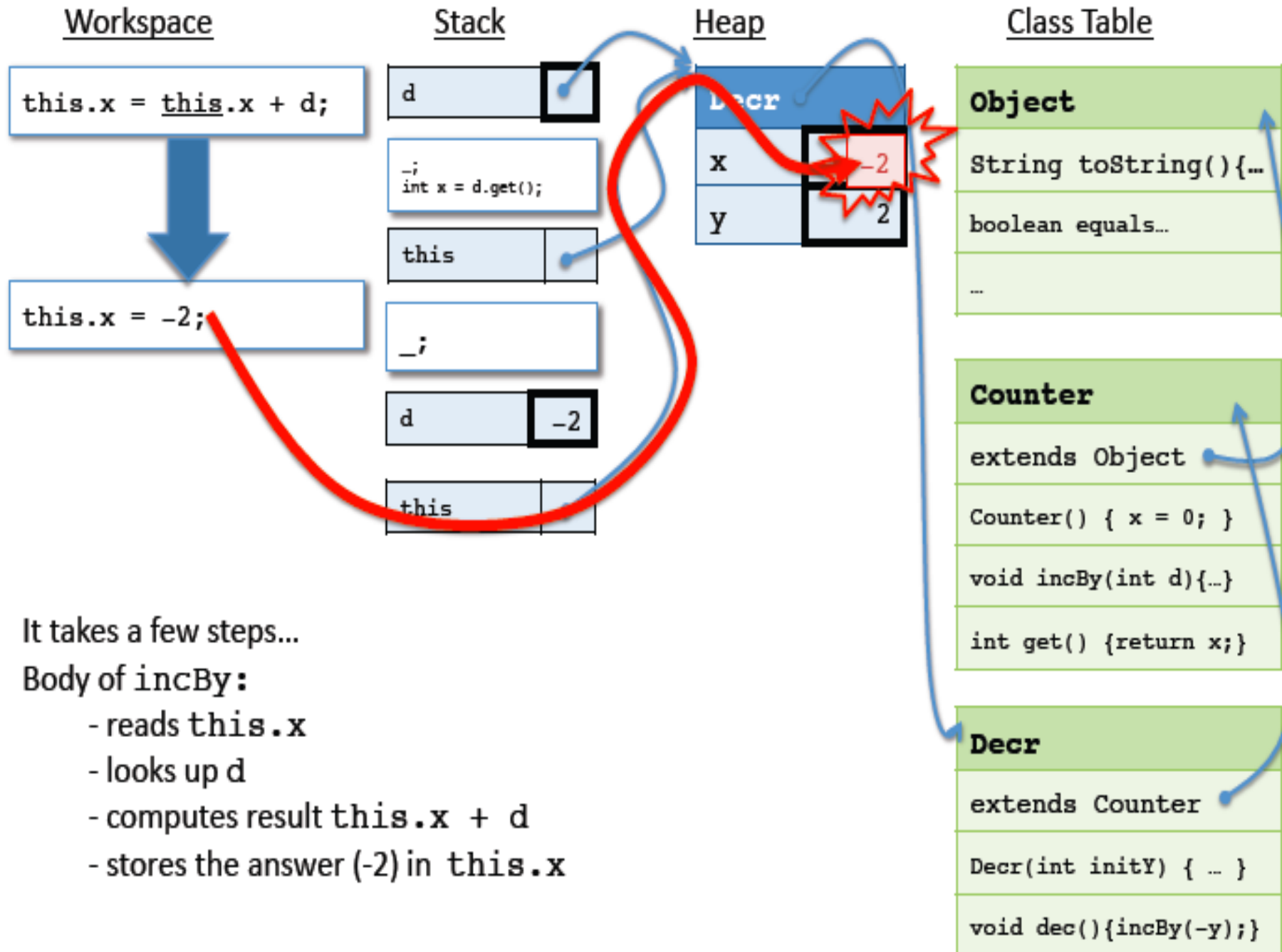
Decr
extends Counter
Decr(int initY) { ... }
void dec(){incBy(-y);}

Invocazione del metodo



Dynamic Dispatch, Again





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`

