PROGRAMMAZIONE 2
5. Tipo statico e dinamico, Dynamic dispatch

Tipo statico

- Il **tipo statico** di una variabile è il tipo della classe (o della interfaccia) che definisce quali oggetti possono essere legati a quella variabile
- Esempio
  - `public class C { … }`
  - `C c = new C( );`
  - **C è il tipo statico della variabile c**

Tipi statici e dinamici

- il **tipo statico** di una espressione è il tipo che descrive il valore calcolato dall'espressione solamente in base alla struttura testuale della espressione (senza valutarla)
- Il **tipo dinamico** di un oggetto è il tipo della classe di cui l'oggetto è istanza

statico vs. dinamico

- La presenza della nozione di **ereditarietà** fa emergere chiaramente questa nozione
- Il **tipo dinamico** di una variabile o di una espressione è sempre un sotto-tipo del **tipo statico**

Esempio: tipi e gerarchia

```
public class Person{ …… }

public class Student  extends Person{ … }

public class Staff extends Person{ … }
```

```
Student p = new Student( );
Staff c = new Staff ( );
Person s1 = p;   Linea A
Person s2 = c;   Linea B
s2 = p;          Linea C
```

Esempio:Tipi e gerarchia

```
public class Person{ …… }

public class Student  extends Person{ … }

public class Staff extends Person{ … }
```

```
Student p = new Student( );
Staff c = new Staff ( );
Person s1 = p;   Linea A
Person s2 = c;   Linea B
s2 = p;          Linea C
```

Quale è il tipo statico
di s1 alla Linea A?

Quale è il tipo dinamico
di s1 alla Linea A
dopo l'assegnamento?

Esempio:Tipi e gerarchia

```
public class Person{ …… }

public class Student  extends Person{ … }

public class Staff extends Person{ … }
```

```
Student p = new Student( );
Staff c = new Staff ( );
Person s1 = p;   Linea A
Person s2 = c;   Linea B
s2 = p;          Linea C
```

Quale è il tipo statico
di s1 alla Linea A?
Person

Quale è il tipo dinamico
di s1 alla Linea A
dopo l'assegnamento?
Student

Esempio:Tipi e gerarchia

```
public class Person{ …… }

public class Student  extends Person{ … }

public class Staff extends Person{ … }
```

```
Student p = new Student( );
Staff c = new Staff ( );
Person s1 = p;   Linea A
Person s2 = c;   Linea B
s2 = p;          Linea C
```

Quali sono i tipi dinamici di s2?

Staff alla Linea B
Student alla Linea C

Esempio:Tipi e gerarchia

```
public class Person{ …… }

public class Student  extends Person{ … }

public class Staff extends Person{ … }
```

```
public Person  prova(Person s) { return s; }
```

```
Student p = new Student( );
Staff c = new Staff ( );
Person s1 = prova(p);
```

Quale è il tipo statico di prova(p)?
Person

Quale è il tipo dinamico di prova(p)?
Student

Dynamic Dispatch

- La dichiarazione di una variabile non determina in maniera univoca 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 (in un unico file)

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

x.start( );//Quale metodo viene invocato?
// Quello di A o quello di B?
      }
}
class A
    public void start( )( )
{System.out.println("Inside start method of A");
    }
}
class B extends A {
    public void start( ) {
 System.out.println("Inside start method of B");
    }
```

Intuizione

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

Un Esempio

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

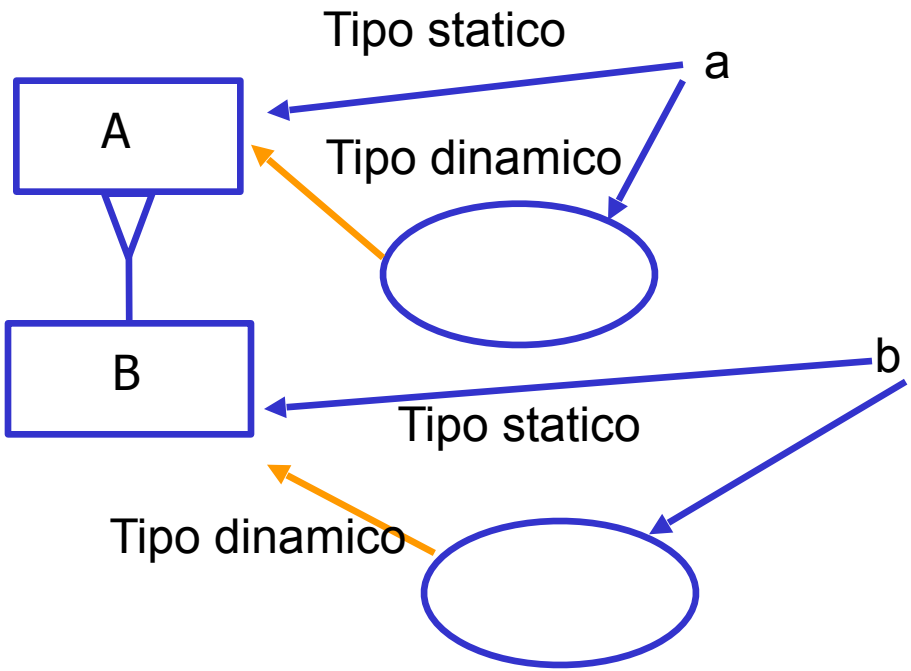
Viene invocato il metodo della classe A

Viene invocato il metodo della classe B

Viene invocato il metodo della classe B

Dynamic Dispatch

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

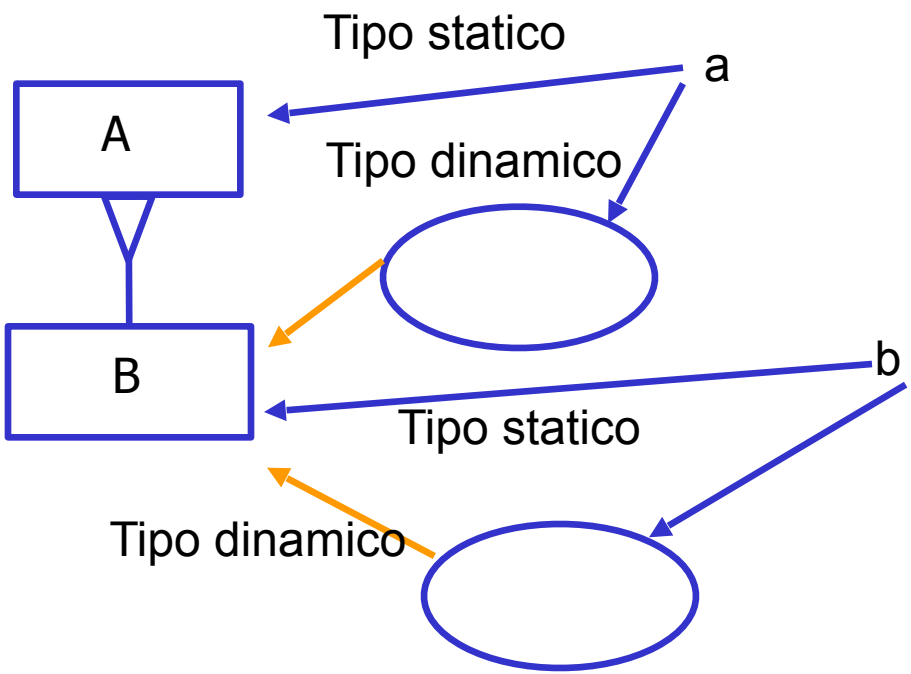



```
A a = new A ( );
B b = new B ( );

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;
a.m( );
```

Tipo statico di a è A
Tipo dinamico di a è B

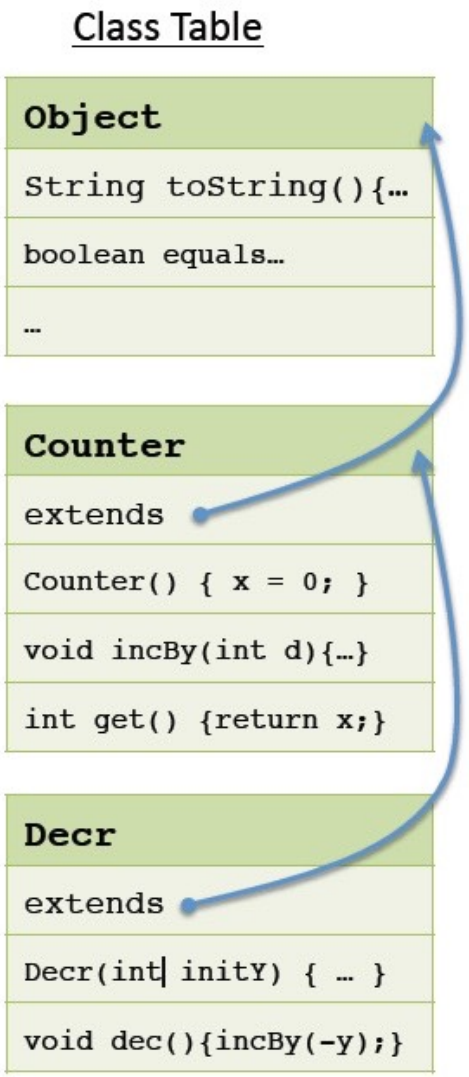
Static vs dynamic

- 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

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 puntatore 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 dynamic 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 {
   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);  }
}
```

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

Animazione dell’esecuzione

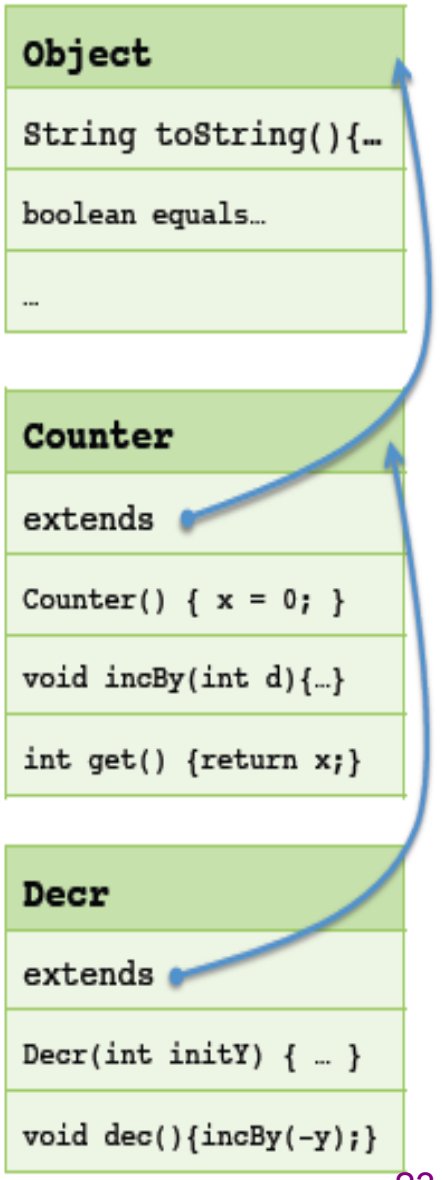
Workspace

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

Stack

Heap

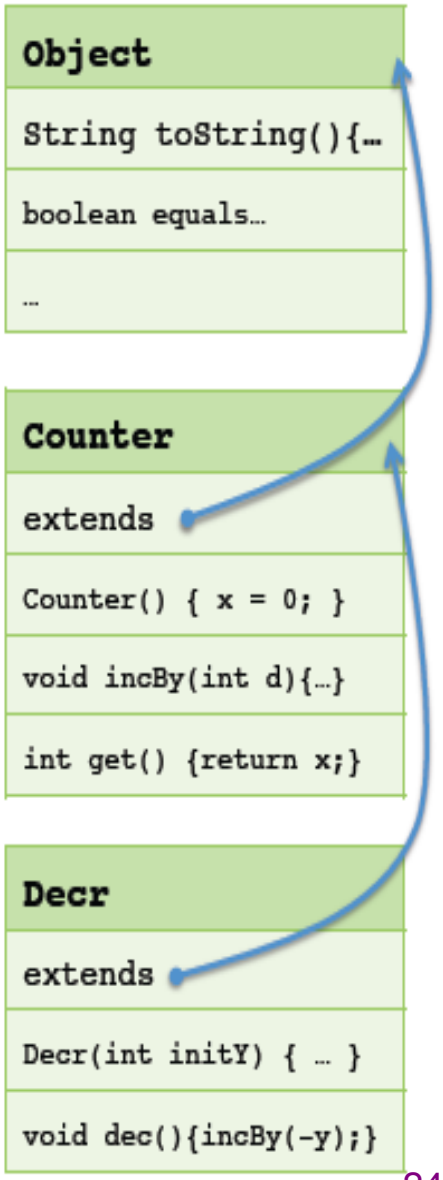
Class Table

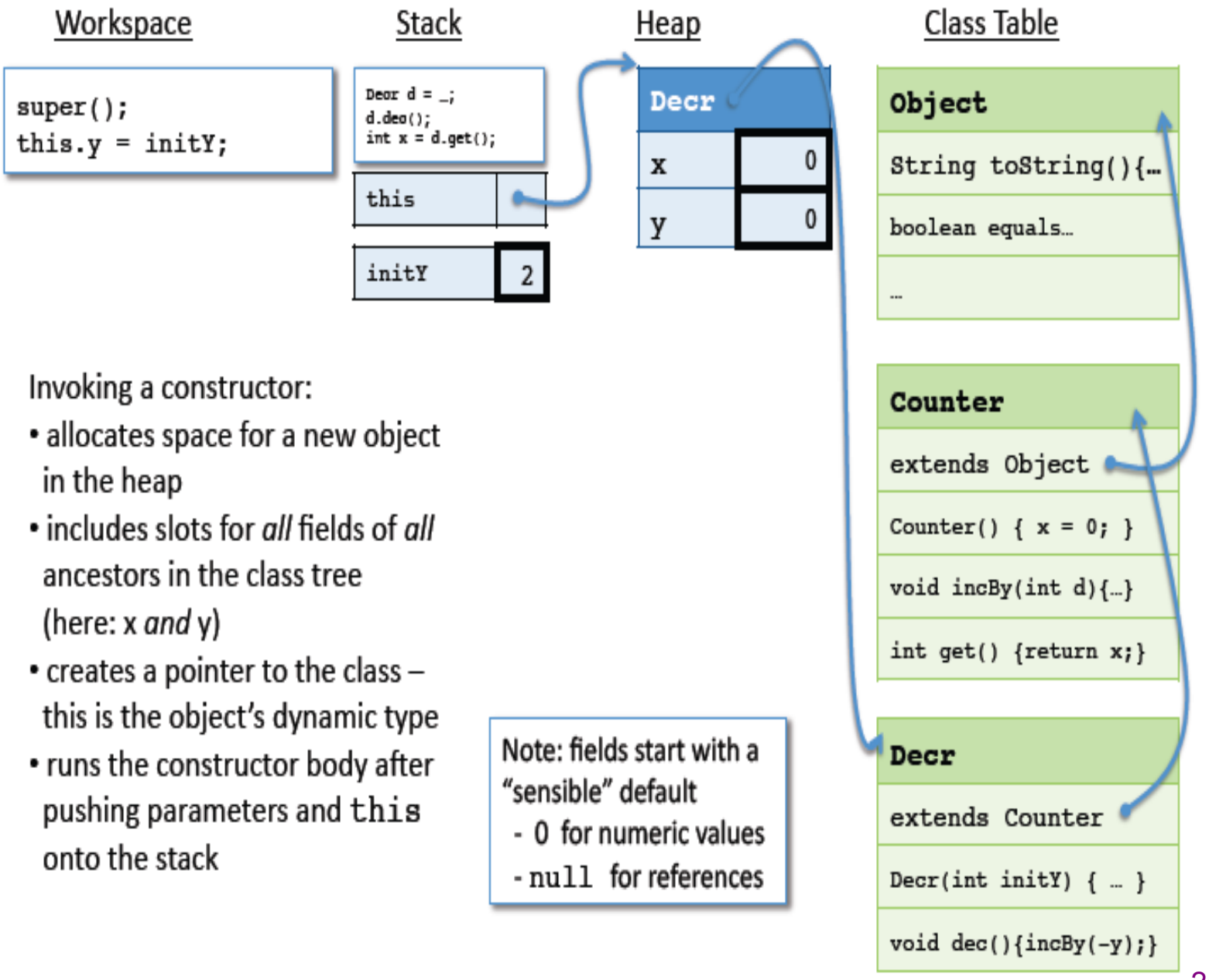


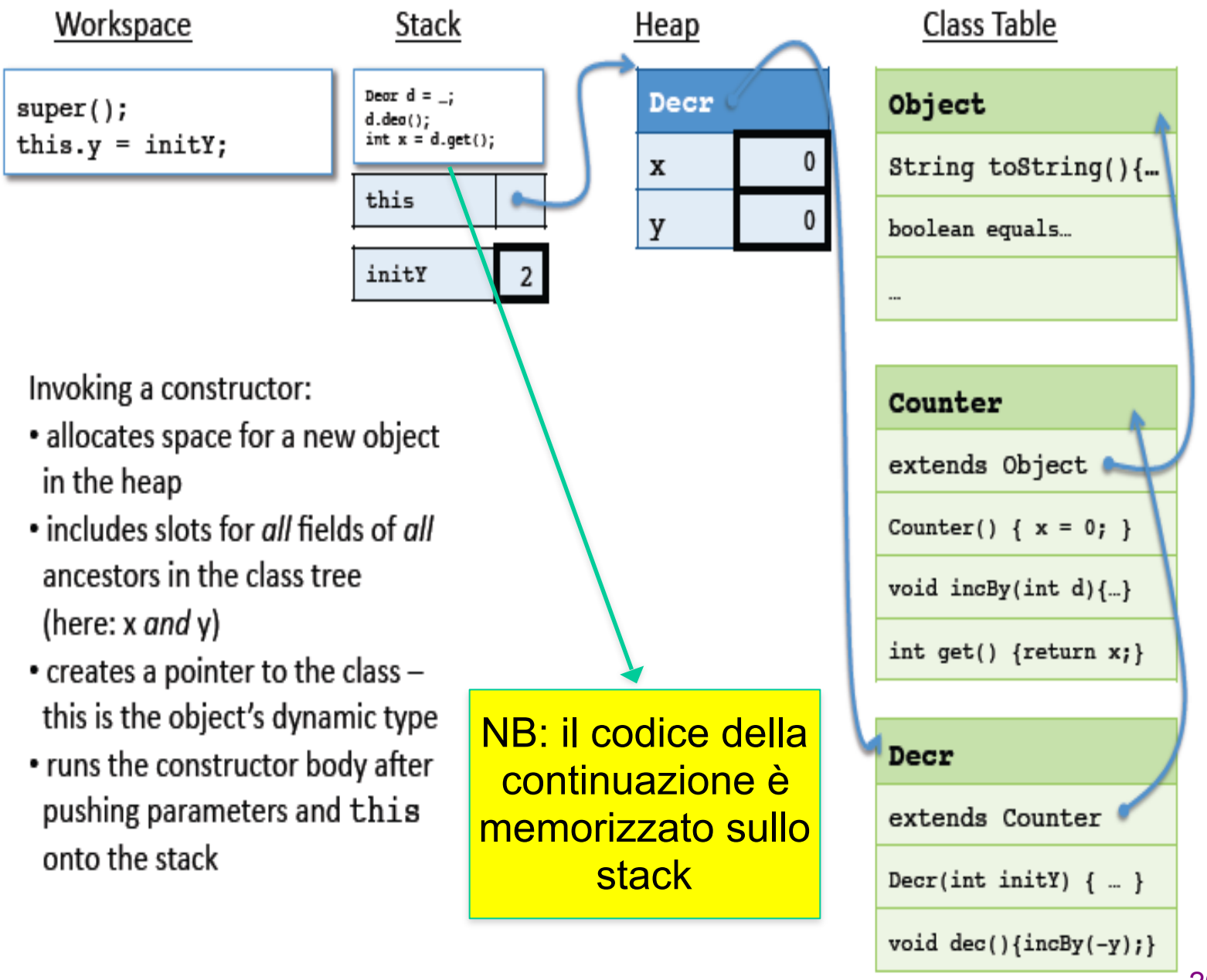
Workspace | Stack | Heap | Class Table

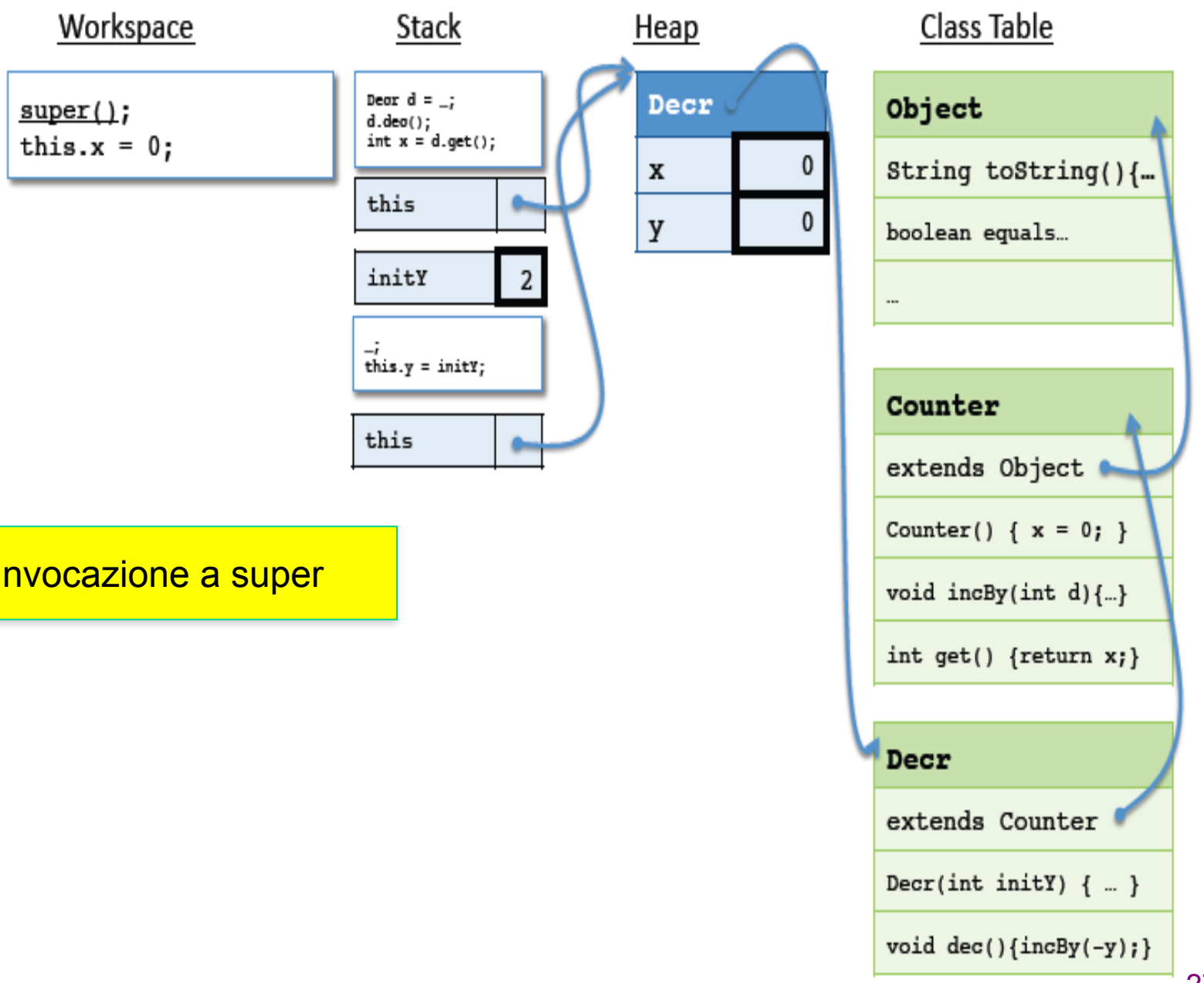
```
Decr d = new Decr(2);
d.dec();
int x = d.get();
```

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



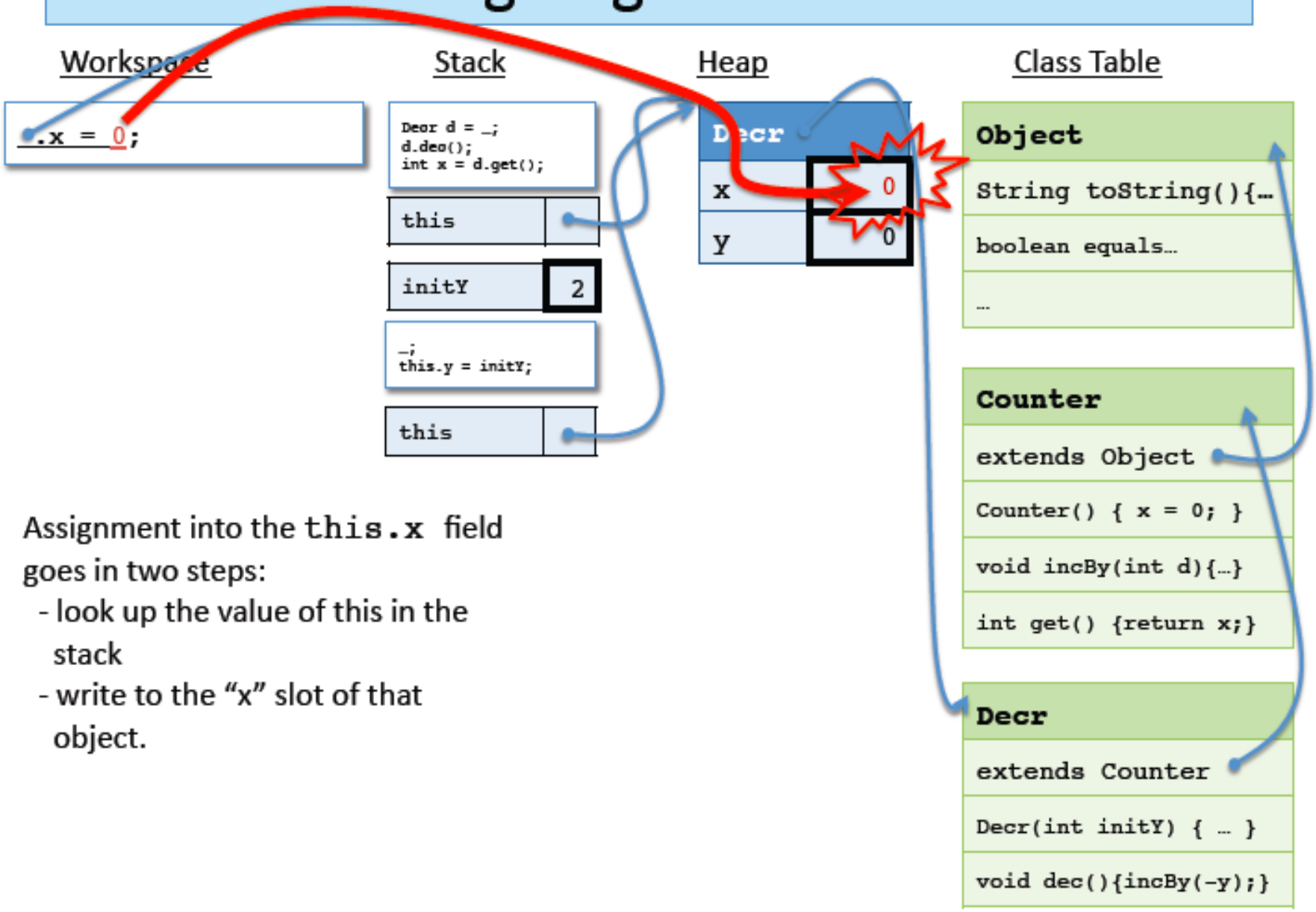

Workspace
super();
this.y = initY;
Stack
Decr d = _;
d.deo();
int x = d.get();
this
initY
2
Heap
Decr
x
0
y
0
Class Table
Object
String toString(){…
boolean equals…
…
Counter
extends Object
Counter() { x = 0; }
void incBy(int d){…}
int get() {return x;}
Decr
extends Counter
Decr(int initY) { … }
void dec(){incBy(-y);}
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;
Decr d = _;
d.deo();
int x = d.get();
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
NB: il codice della continuazione è memorizzato sullo stack



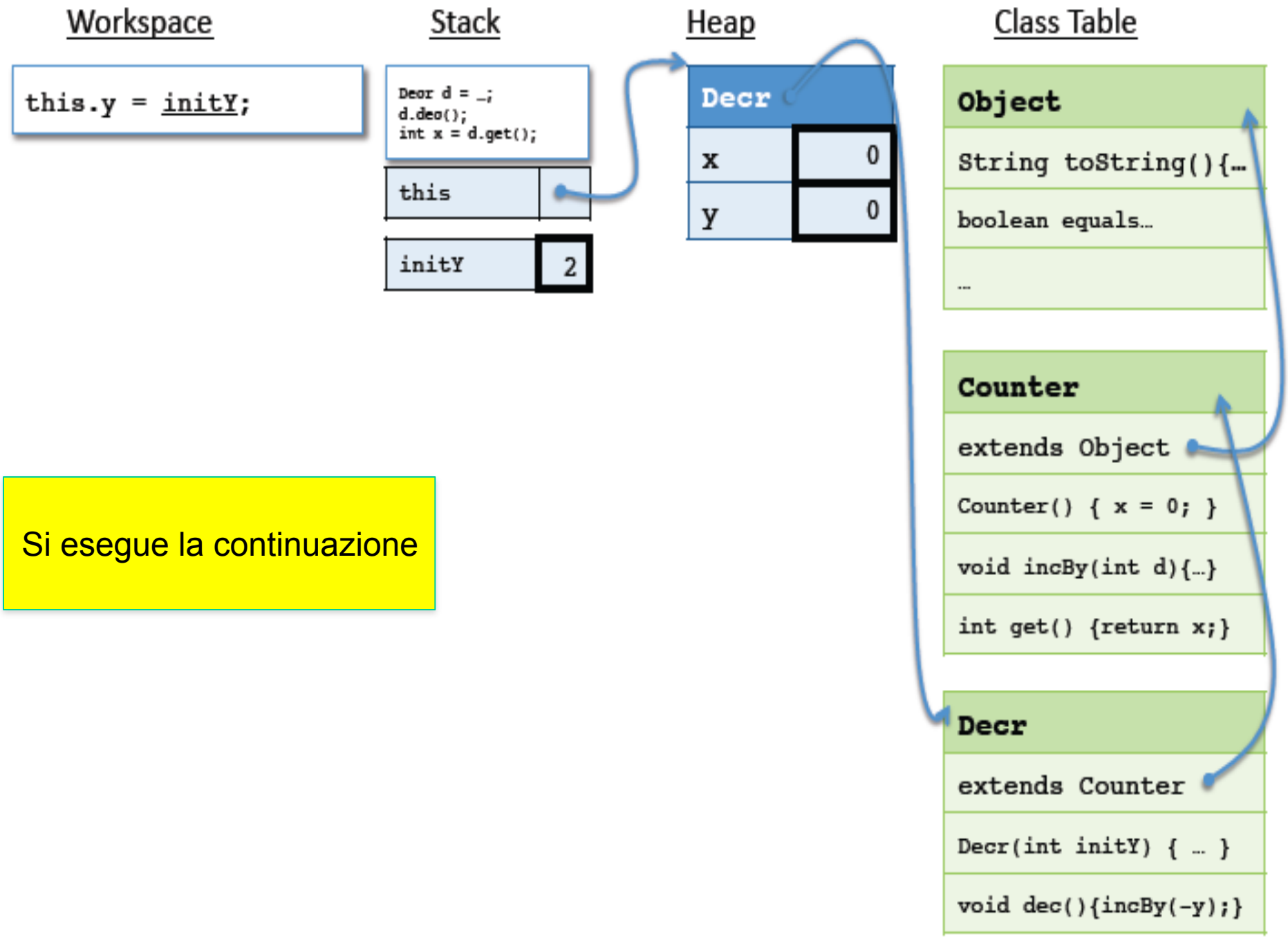
Invocazione a super

Assigning to a Field

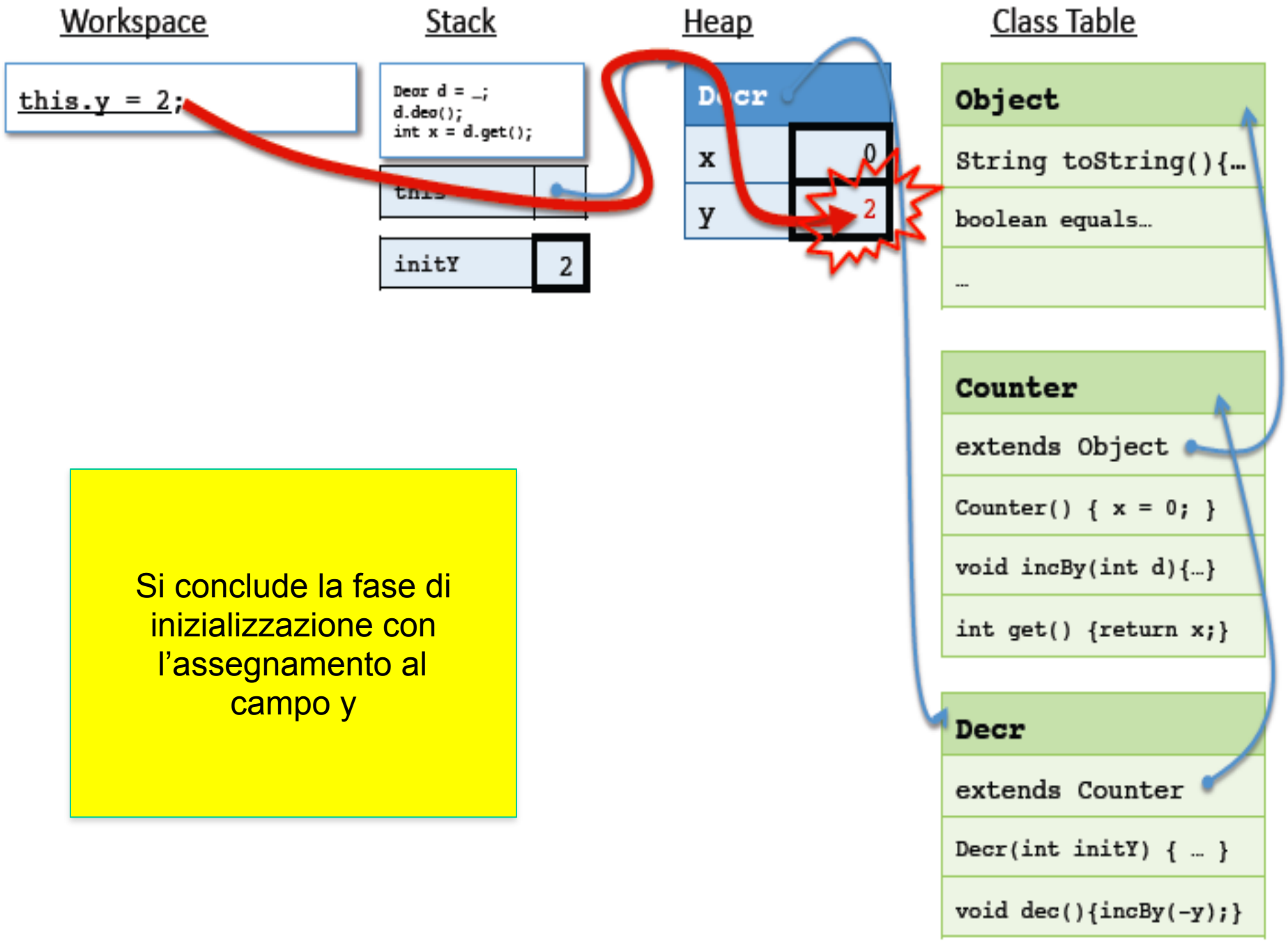


Assignment into the `this.x` field goes in two steps:

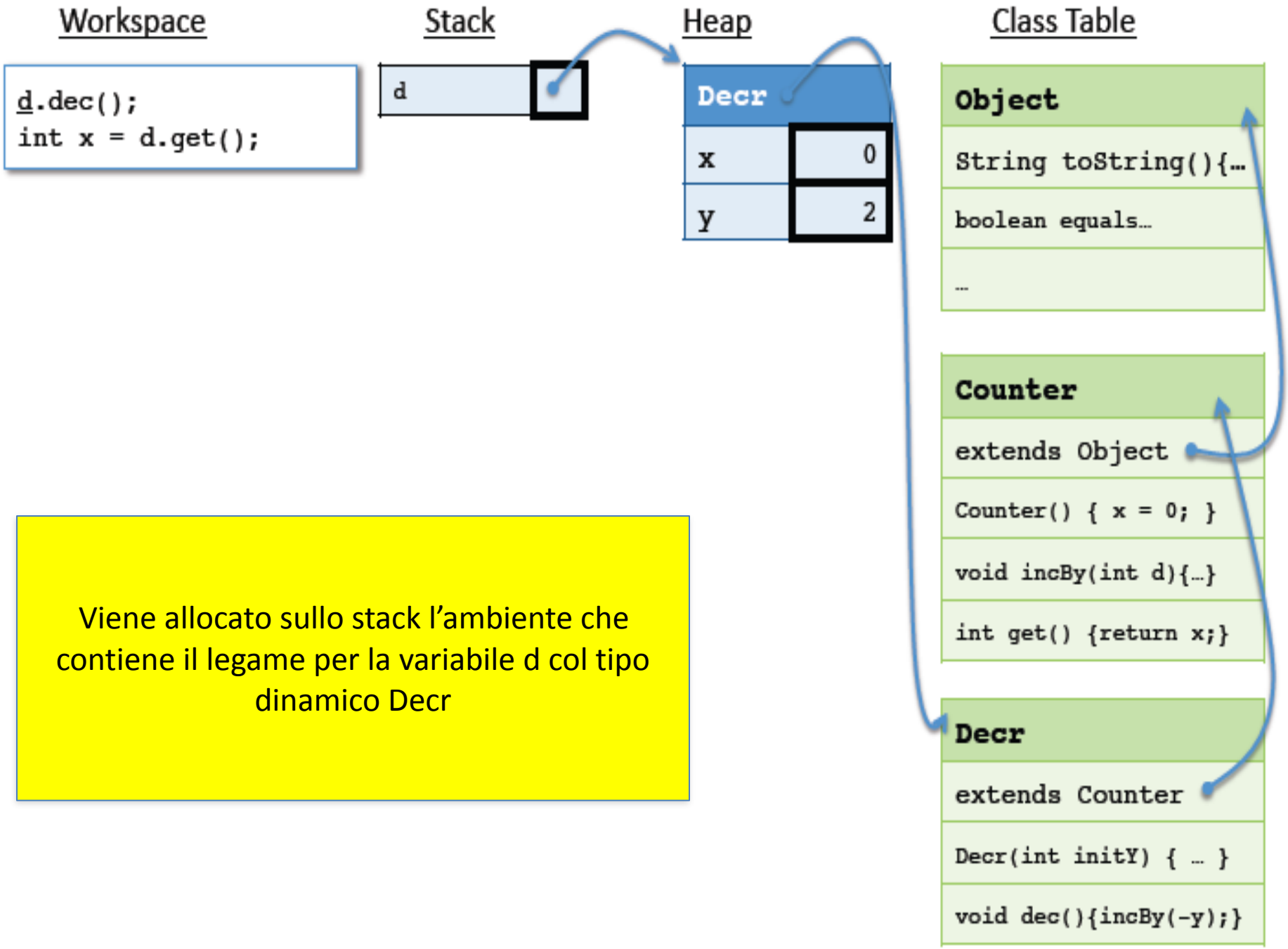
- look up the value of this in the stack
- write to the "x" slot of that object.



Si esegue la continuazione

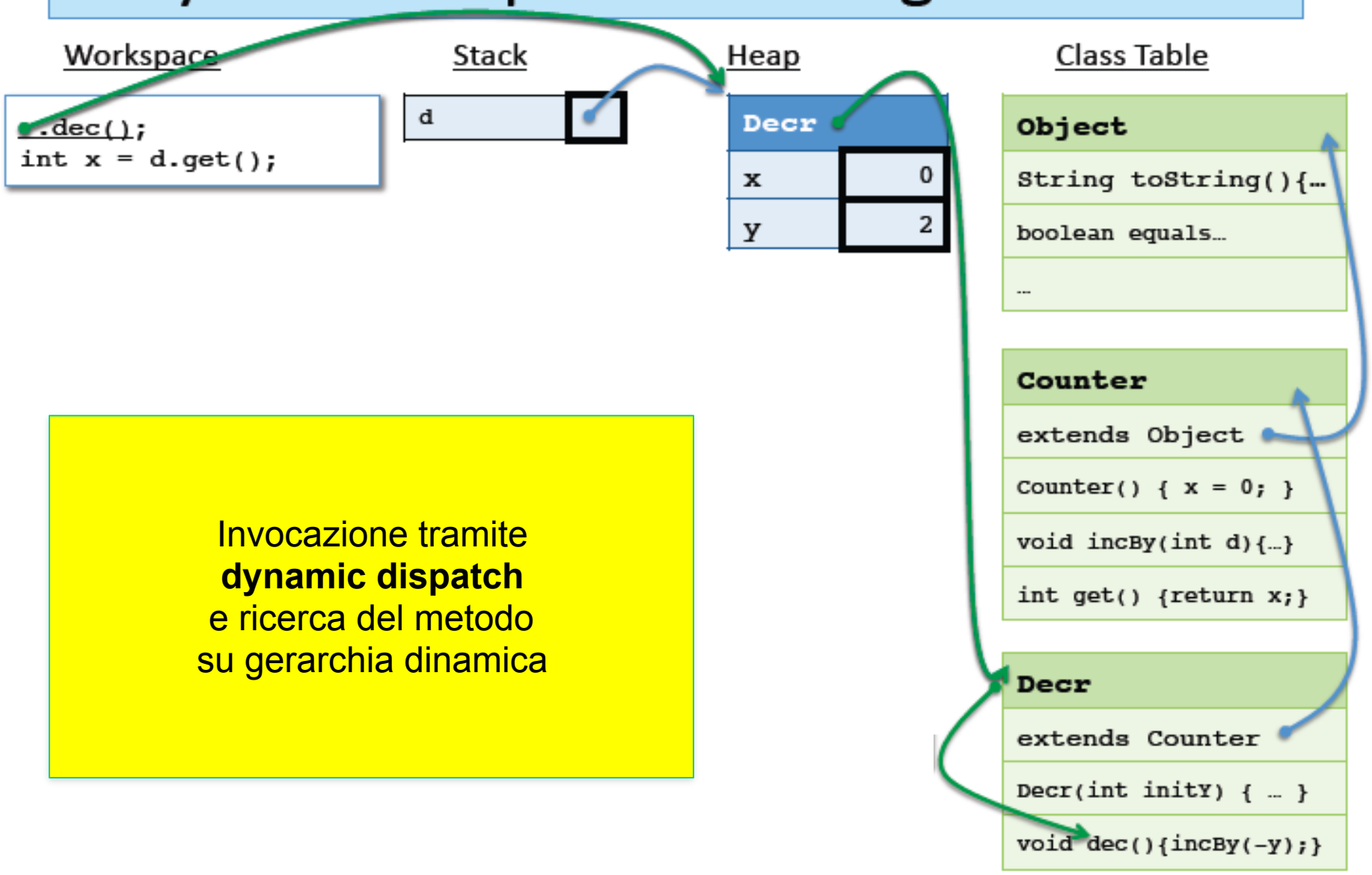


Si conclude la fase di inizializzazione con l'assegnamento al campo y

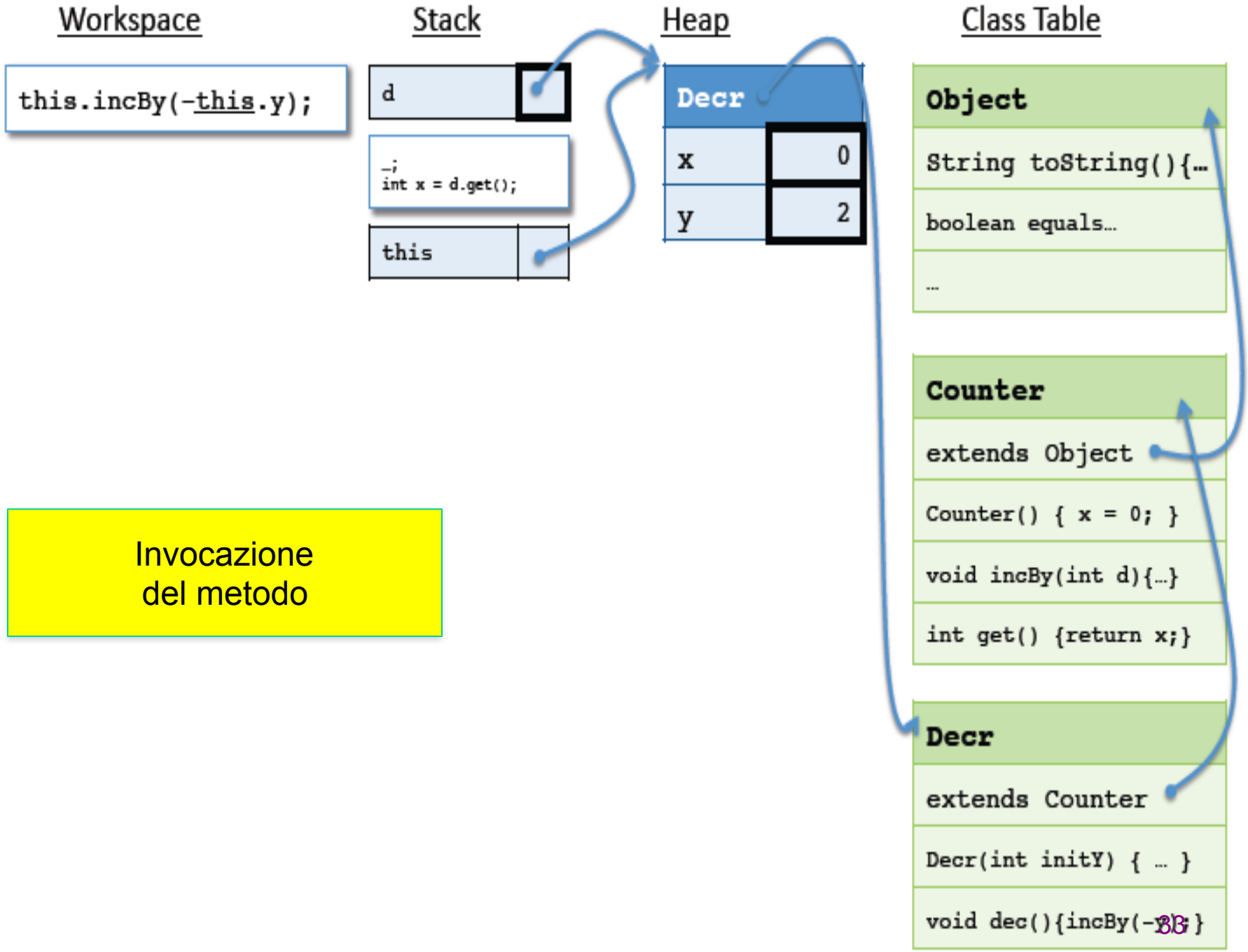


Viene allocato sullo stack l'ambiente che contiene il legame per la variabile d col tipo dinamico Decr

Dynamic Dispatch: Finding the Code

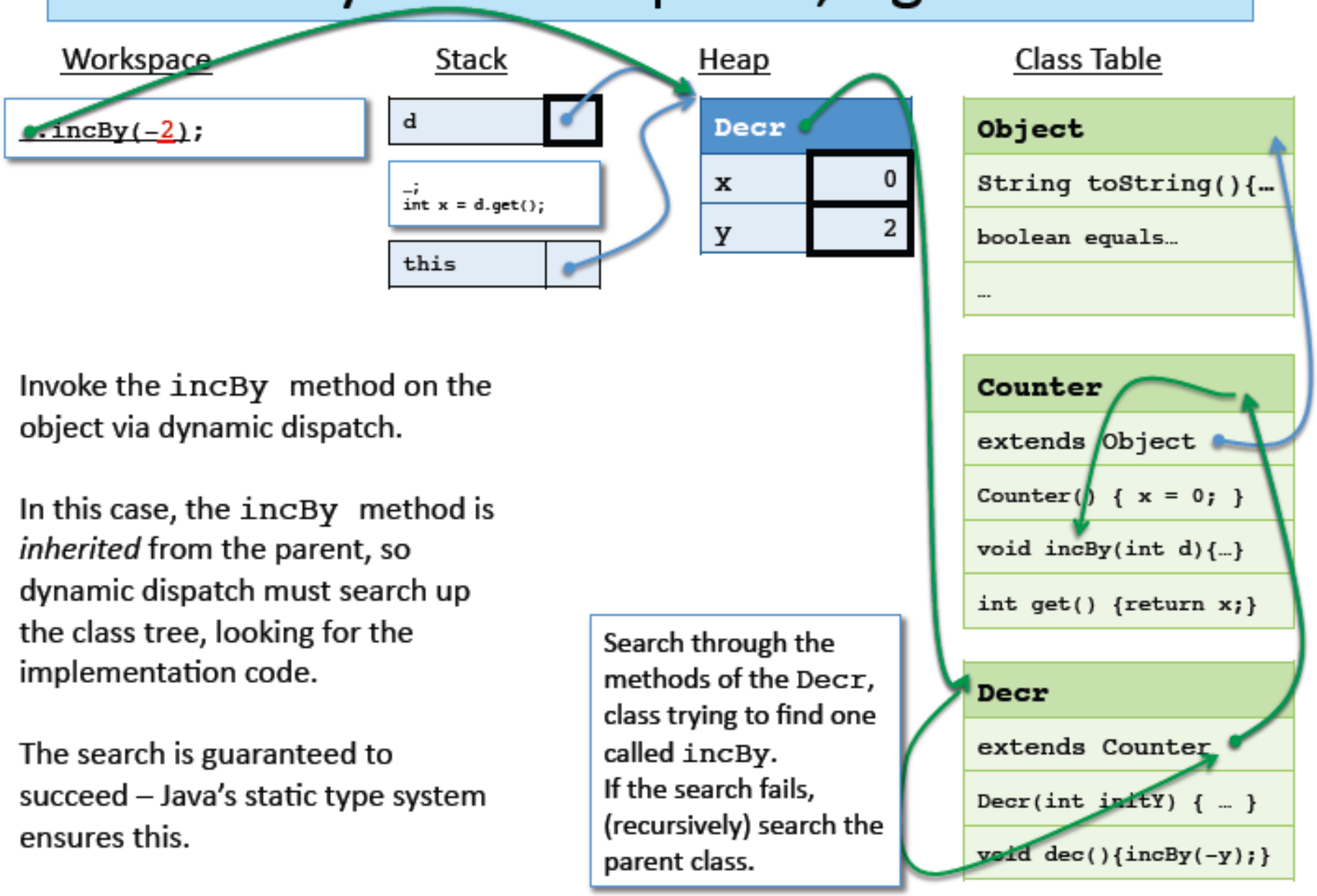


Invocazione tramite
dynamic dispatch
e ricerca del metodo
su gerarchia dinamica



Invocazione del metodo

Dynamic Dispatch, Again

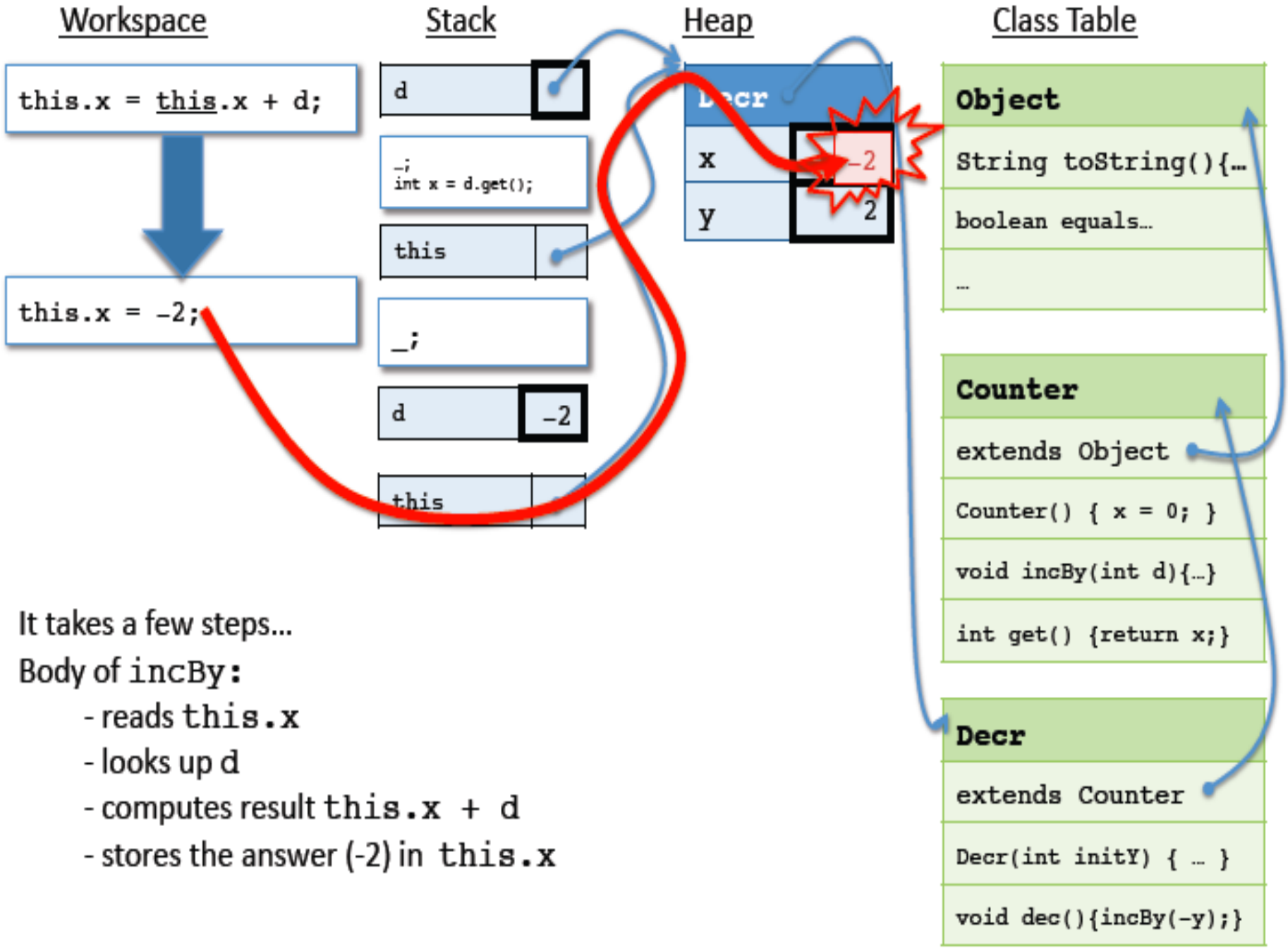


Invoke the `incBy` method on the object via dynamic dispatch.

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.

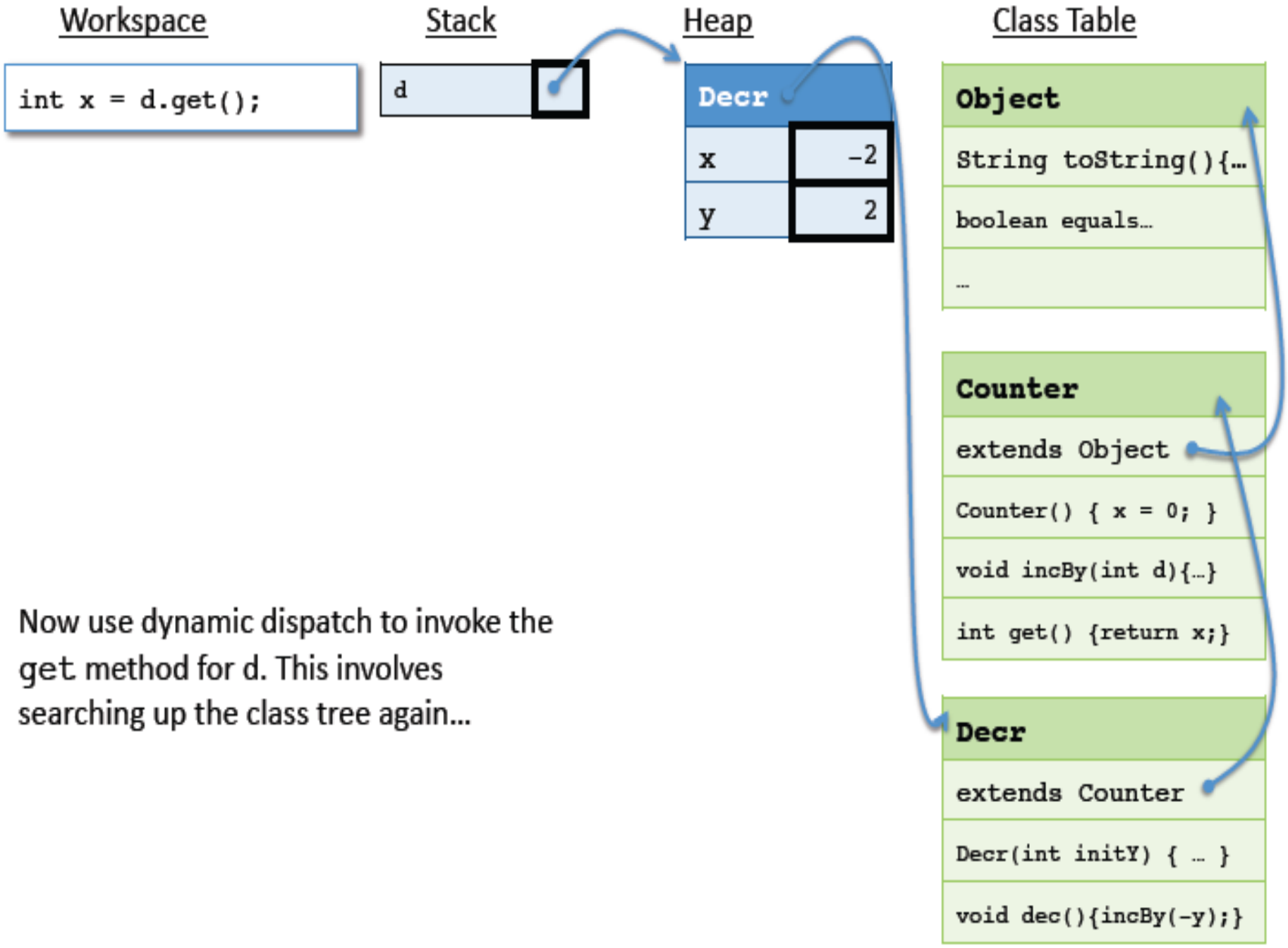
Search through the methods of the `Decr`, class trying to find one called `incBy`.
If the search fails, (recursively) search the parent class.



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`



Now use dynamic dispatch to invoke the `get` method for d. This involves searching up the class tree again...

Esercizi: tipo statico e dinamico

```
public class A {
  public void m1()
         { …. }
  public void m2( ){……}
}

public class B extends A{
  public void m1()
         { …. }
  public void m3(){……}
}

public class C extends B{
public void m1() {}
public void m2(){……}
}
```

Esercizio

```
public static void main(String[] args) {
  A b = new B();
  b.m3();
  A c=new C();
  c=b;
  a.m1(); }
```

Il frammento Java supera o meno la fase di compilazione?

```
class A {
  void f() {
    System.out.println("A f");
  }
}
class B extends A {
  void f() {
super.f();
    System.out.println("B f");
  }
}
public static void main(String[] args) {
  B b = new B();
  b.f();
  A a = b;
  a.f(); }
```

Descrivere il risultato osservabile dell'esecuzione del metodo main.